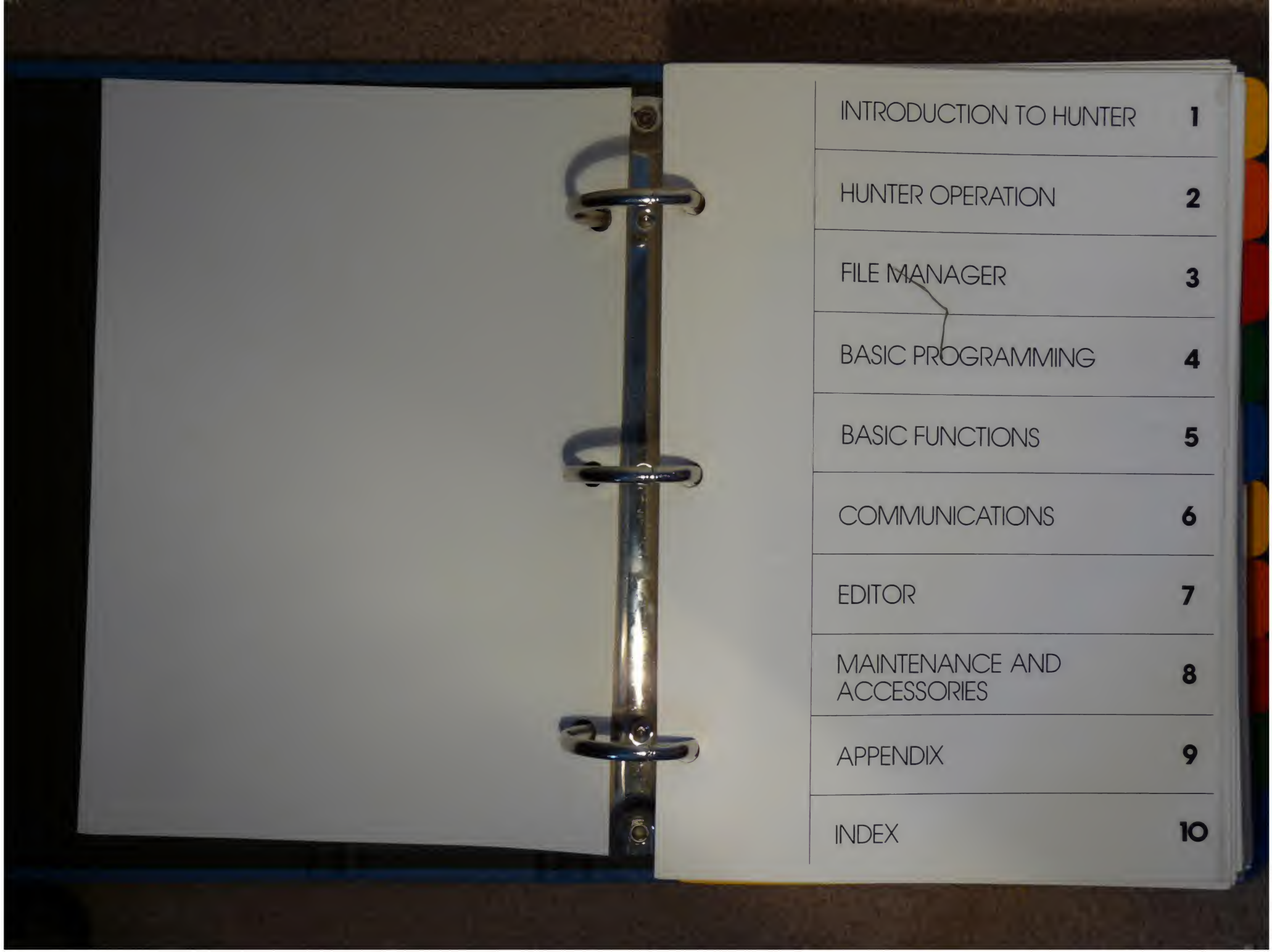
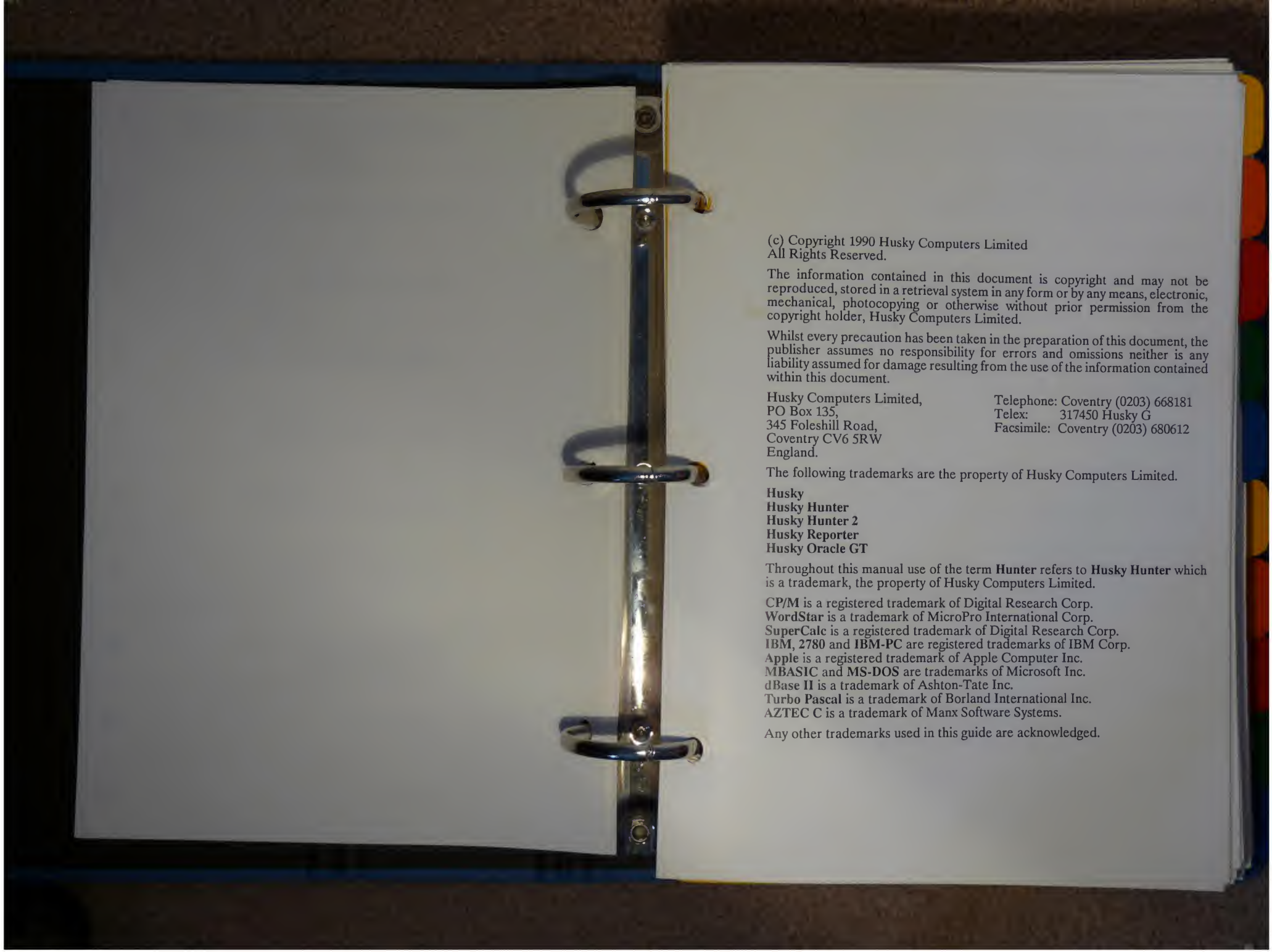
HUSKY HUNTER

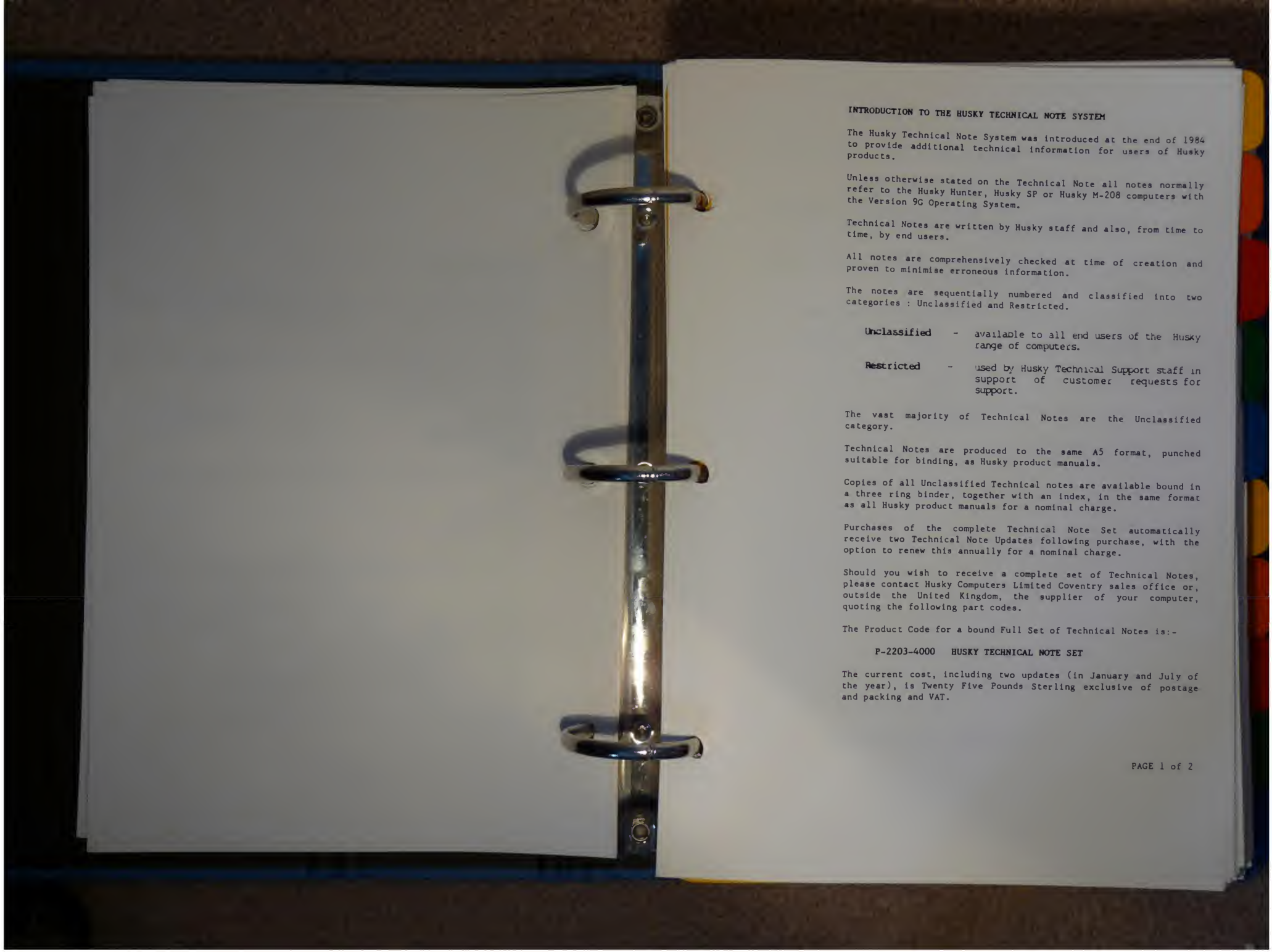
Operations Manual





Husky Computers Limited,
PO Box 135,
345 Foleshill Road,
Coventry CV6 5RW
England.

Telephone: Coventry (0203) 668181
Telex: 317450 Husky G
Facsimile: Coventry (0203) 680612



INTRODUCTION TO THE HUSKY TECHNICAL NOTE SYSTEM

The Husky Technical Note System was introduced at the end of 1984 to provide additional technical information for users of Husky products.

Unless otherwise stated on the Technical Note all notes normally refer to the Husky Hunter, Husky SP or Husky M-208 computers with the Version 9G Operating System.

Technical Notes are written by Husky staff and also, from time to time, by end users.

All notes are comprehensively checked at time of creation and proven to minimise erroneous information.

The notes are sequentially numbered and classified into two categories : Unclassified and Restricted.

Unclassified - available to all end users of the Husky range of computers.

Restricted - used by Husky Technical Support staff in support of customer requests for support.

The vast majority of Technical Notes are the Unclassified category.

Technical Notes are produced to the same A5 format, punched suitable for binding, as Husky product manuals.

Copies of all Unclassified Technical notes are available bound in a three ring binder, together with an index, in the same format as all Husky product manuals for a nominal charge.

Purchases of the complete Technical Note Set automatically receive two Technical Note Updates following purchase, with the option to renew this annually for a nominal charge.

Should you wish to receive a complete set of Technical Notes, please contact Husky Computers Limited Coventry sales office or, outside the United Kingdom, the supplier of your computer, quoting the following part codes.

The Product Code for a bound Full Set of Technical Notes is:-

P-2203-4000 HUSKY TECHNICAL NOTE SET

The current cost, including two updates (in January and July of the year), is Twenty Five Pounds Sterling exclusive of postage and packing and VAT.

The Product Code for Updates to the full Set of Technical Notes is:-

P-2203-4001 HUSKY TECHNICAL NOTE UPDATES

The current annual charge, covering two updates (in January and July of the year), is Twenty Five Pounds Sterling exclusive of postage and packing and VAT.

Note: Any orders for Technical Note Sets or Updates should include the name and address of the person within the ordering organisation to whom they should be sent.

Husky Computers Limited
P.O. Box 135
345 Foleshill Road
COVENTRY
CV6 5RW
England

Telephone: (0203) 668181

Telex: 317450 Husky G

February 1986

HUNTER

OPERATIONS MANUAL

REVISION: VER.VO9F SEPTEMBER 1984

PREFACE

The HUNTER microcomputer is a self contained and truly portable system. Its built-in operating system contains an accurate emulation of the world-standard CP/M operating system and virtual floppy disk drive, implemented in RAM memory. Other built-in features include a powerful BASIC interpreter, a terminal emulation package, a flexible text editor with full-screen facilities based on the popular Wordstar and a comprehensive range of communications protocols. User memory is shared between work areas, program and data files.

REPORTED 'BUGS'

Some of the reported 'bugs' which existed in the system have been corrected.

These are as follows:

a) The format of the KILL statement in Basic has been extended to include the form Kill A$,B$.... The use of 'name strings' in the NAME statement has also been included.

b) The calls via Basic are also functioning correctly. This enables the call for the status of the serial port to be used again in place of the manual peeking of the communications pointers.

c) The reported problems with IF THEN ELSE have been corrected.

d) Machine code calls from Basic are also functioning.

e) Line feeds are no longer echoed to the screen during LLOAD, LPRINT etc.

f) The REM and ' have been separated out into separate tokens.

g) Communications timeouts have been introduced.

h) It is now possible to SAVE and LOAD both ASCII and Basic tokenised files from the Basic interpreter.

i) In some versions of operating system, complex string expressions did lead to error. This has been corrected in the new version.

j) In previous versions, the wand returned ',' in place of '-' in CODE39.

There are still, however, some idiosyncrasies which as yet are not corrected in this release of the operating system, but these will be corrected when further releases are produced.

a) WHILE-WEND requires that the WHILE is at the start of a line in Basic.

b) There are some problems with the VAL statement in basic, e.g. VAL(".005") gives 0 not 0.005.

c) The SRCH statement does not function correctly with null length string.

d) PRINT USING is not incorporated in this version of operating system.

Your co-operation in reporting faults in the system is very much appreciated. Any discrepancies in the operating system or the associated documentation should be reported to the Husky office either by telephone or in writing.

Users who have any strong feelings regarding any lack of facilities in the HUNTER would also be welcome to put forward their ideas. Although we cannot promise to include the idea into the system, it will be given serious consideration for inclusion in a later version if viable.

There have been some updates to the operating system which are not dealt with in the current release of the HUNTER manual. the following is a description of the new changes.

WAND

In place of the wand menu on earlier versions, a new verb has been added to the BASIC in order to simplify the wand type selection. This new verb is WAND.

```
setting wand=0      selects Code 39  bartype
setting WAND=1      selects EAN 8/13 bartype
```

WAND may also be used as parts of expressions, e.g:

```
PRINT WAND
PRINT WAND*5
```

The location may be poked direct (BARTYPE)

TERMINAL MODE

Terminal mode has now been provided with function keys in order to enable the user to exit and also to set the communications parameters and return back to terminal mode direct.

COMMUNICATION PARAMETERS

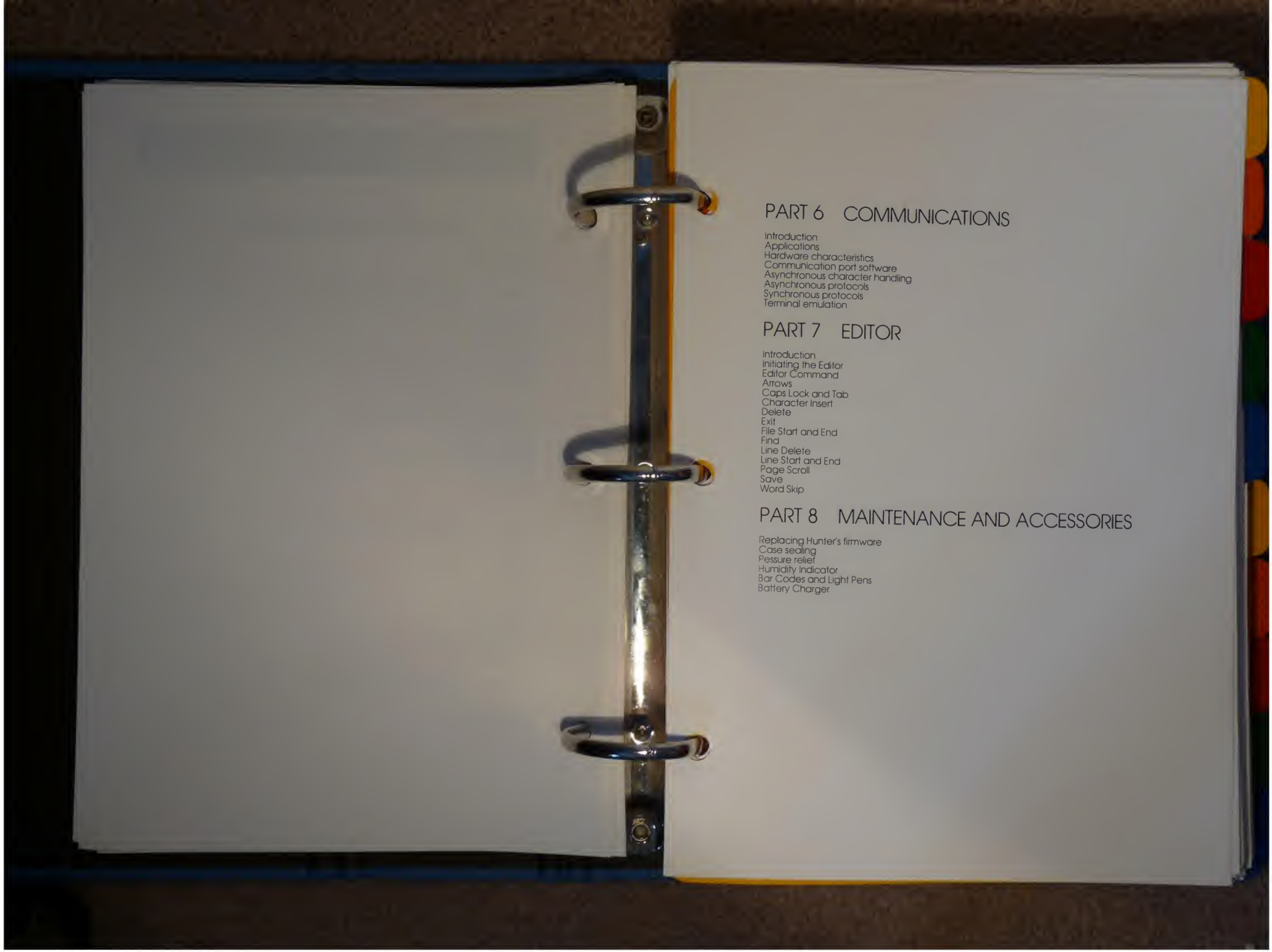
There has been an addition to the communications parameters in order to provide a timeout option on both receive and transmit communications. These values can be set from 10 seconds to 1 minute in intervals of 10 secs. Unlike the protocol timeouts, they can be disabled by selecting 'no' in the menu.

The receive timeout is activated when the HUNTER requests an input from the port. If an input is not received within the specified time, then a communications error '8' is given. The user may elect to try again by pressing 'X' where upon the HUNTER will wait for an input again for the selected time and timeout again if nothing is received.

The transmit timeout is activated when the HUNTER tries to transmit but is prevented from doing so by the various handshake lines. If after the specified time the HUNTER is still not able to transmit, then a communications error '9' will occur. By pressing 'X' the HUNTER can be made to re-attempt the transmission.

Both transmit and receive timeouts can be trapped by use of the BASIC 'on comms goto' statement and the error number returned in location COMERR.

CONTENTS



PART 6 COMMUNICATIONS

Introduction
Applications
Hardware characteristics
Communication port software
Asynchronous character handling
Asynchronous protocols
Synchronous protocols
Terminal emulation

PART 7 EDITOR

Introduction
Initiating the Editor
Editor Command
Arrows
Caps Lock and Tab
Character Insert
Delete
Exit
File Start and End
Find
Line Delete
Line Start and End
Page Scroll
Save
Word Skip

PART 8 MAINTENANCE AND ACCESSORIES

Replacing Hunter's firmware
Case sealing
Pessure relief
Humidity Indicator
Bar Codes and Light Pens
Battery Charger

PART 9 APPENDIX

Hunter specification
ASCII character set
HEX to Decimal conversion
NSC800 Machine code
Demonstration programs
Keyboard memory map
Memory locations
Port allocations
Single bit input port
RS-232 connector
HEX data format
ASCII to EBCDIC conversion
Four octave sound range
Configuring Typical CP/M Programs

PART 10 INDEX

Alpha index
Index to figures
Index to tables


INTRODUCTION TO HUNTER
CONTENTS
1.1 WELCOME TO HUNTER
1.2 USING THE MANUAL
1.3 PAGE FORMATS

WELCOME TO HUNTER

1.1

Fig 1.1 THE HUNTER

Husky HUNTER is probably the smallest and toughest CP/M microcomputer ever made.

It is resistant to moisture, dust, vibration, shock and electro-magnetic interference. But please don't abuse it. Inside the impervious cast aluminium case is some of the most advanced technology money can buy. Don't drop HUNTER needlessly, use it to strike other objects or pile heavy things on top of it.

If you can't find the information you need in this manual or you have any problems with HUNTER, please call us. We've done all we can to make HUNTER a practical and utilitarian tool, not a frustrating incumbrance. We hope you agree. After all, HUNTER is our baby.

The HUNTER team

HUSKY COMPUTERS LTD
P O BOX 135
345 FOLESHILL ROAD
COVENTRY CV6 5RW
ENGLAND

TEL:(0203) 668181
TELEX: 317450 HUSKY G

1.1.3 **HUNTER**

Measuring 21.6 X 15.6 X 3.7 cm (8.5" x 6.2" x 1.3") and weighing less than 1200g (3lb), HUNTER is a completely self contained computer system. HUNTER is sealed against moisture, dust and other hazards.

NEVER attempt to open the case - there are no user serviceable parts inside.

The CP/M emulating operating system, now containing a flexible text editor, has RAM-disc emulation and provides a totally compatible environment for a vast range of commercially available software.

HUNTER's Basic interpreter has been specially developed to be compatible with other popular Basics and to meet the extra demands of a portable system. Special statements, for example to filter input data so that only formats consistent with the program are accepted, simplify user programming.

Communications with instruments or computers, including mainframes, are user programmable. Protocols can be chosen from a wide selection, including a choice of synchronous or asynchronous types.

CP/M compatibility is an important feature. A large range of commercially available software, including packages such as spreadsheets, can now be run on HUNTER with no modifications.

Disk emulation in RAM memory allows standard programs such as WordStar and SuperCalc to run without modification, while files can be exchanged easily.

WordStar is a registered trademark of MicroPro International Corporation.

SuperCalc is a registered trademark of Sorcim.

CP/M is a registered trademark of Digital Research Corp

USING THE MANUAL

1.2.1 This manual is intended to fulfil several functions.

1) As an introduction to the operation, programming and use of HUNTER.

2) As a source of reference for technical data about HUNTER.

3) As the vehicle for achieving HUNTER's full potential in diverse user applications.

It is not intended as a guide for the first time computer or Basic user. For guidance on these subjects, the reader is referred to the many excellent introductory works dealing with microcomputer Basic, now commonly available.

1.2.2 **HOW TO USE THIS MANUAL**

The manual is split into a number of parts. Part 1 is this introduction. The others are:

2) HUNTER Operation:
Description of HUNTER - its File Manager, communications, virtual screen and setting of the internal clock/calandar.

3) File Manager:
Description of HUNTER's File Manager DEMOS (Disk Emulation Operating System) and CP/M compatibility.

4) Basic Programming:
Description of HUNTER Basic Programming techniques, use of the Basic Editor plus listing of error and warning messages.

5) Basic Functions:
Detailed explanation of the resident HUNTER Basic interpreter commands.

6) Communications:
Full description of HUNTER's flexible serial data communications facilities.

7) HUNTER Text Editor:
Instruction in the use of the text editing facility within HUNTER.

8) Maintenance and Accessories:
Instructions for case sealing, pressure relief etc., and information of accessories available for Hunter.

9) Appendix:
Specification, code conversion tables, etc.

10) Index:
Alphabetic index to pages.

The contents of each section are listed on the first page following the divider.

PAGE FORMATS

1.3

Each page in this manual is laid out on a standard format. Sections are identified in the left margin by section numbers e.g. 1.3, meaning Part 1, section 3.

The top of each page contairs the part name to the left and the first section number of the page to the right. At the foot appears HUNTER's operating system revision number (to which this manual refers) to the left and the page number to the right.

The page number is in two parts. The first number refers to the part of the manual being studied, the second to the page number referenced from the start of each part.

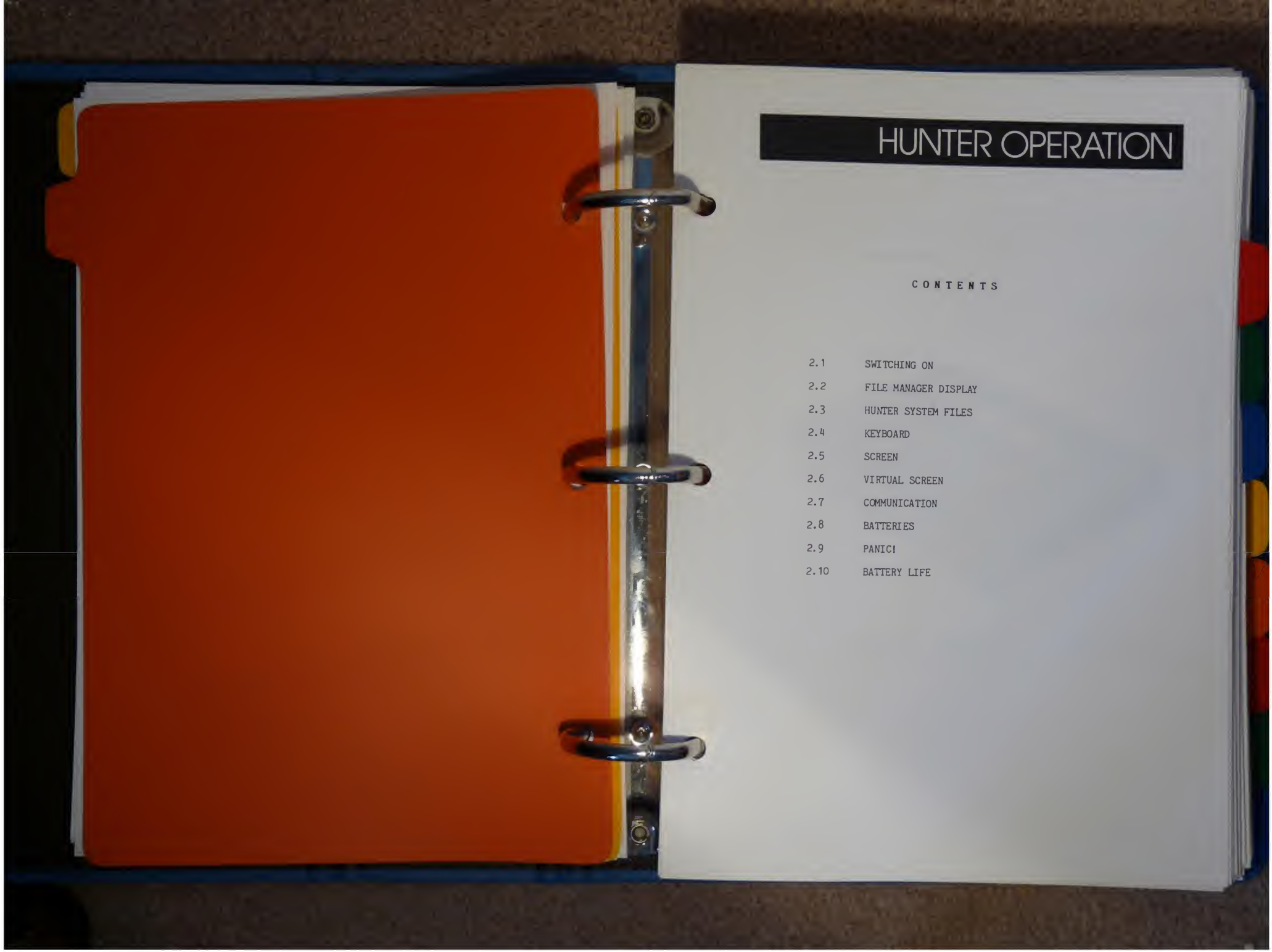
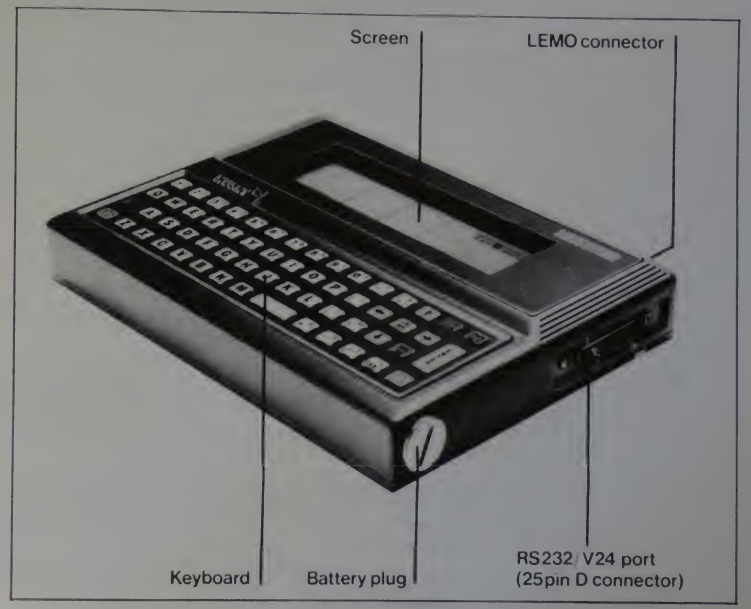

HUNTER OPERATION
CONTENTS
2.1 SWITCHING ON
2.2 FILE MANAGER DISPLAY
2.3 HUNTER SYSTEM FILES
2.4 KEYBOARD
2.5 SCREEN
2.6 VIRTUAL SCREEN
2.7 COMMUNICATION
2.8 BATTERIES
2.9 PANIC!
2.10 BATTERY LIFE

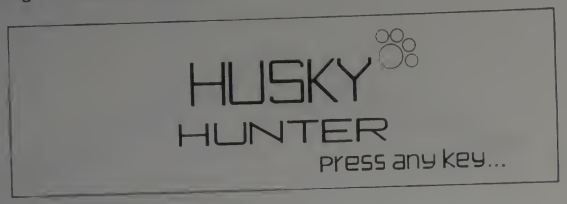
Fig 2.1 HUNTER LAYOUT



SWITCHING ON

2.1.1 To switch HUNTER on press the key marked PWR at the top right hand corner of the keyboard. HUNTER should respond with a 'beep'. If not, check that the batteries have been installed or need to be replaced. See section 2.8, BATTERIES. If all is well the following 'Welcome Message' should be displayed:-

Fig 2.2 'WELCOME MESSAGE'



If all is not well and the above display does not appear, it could be for one of the following reasons.

2.1.1.1 **Display needs adjustment:** The Liquid Crystal Display (LCD) used in HUNTER can vary its angle of display to suit various operating positions. This is obtained by pressing CTL/FN (control/function) and either the cursor up or cursor down keys until the desired display angle is found.

2.1.1.2 **Auto start program loaded:** A program has been entered into HUNTER to automatically run when ever it is switched on. If this is the case the operator should be prompted by HUNTER or have been informed what to do next.

If neither of the above produce a response, check the PANIC section 2.9 for further information.

2.1.2 **SELECTING THE FILE MANAGER**

Having obtained the above 'Welcome Message' HUNTER's File Manager is entered by pressing any key. The screen will then display HUNTER's various system programs, any of which may be selected by operation of the appropriate function key.

FILE MANAGER DISPLAY

2.2 HUNTER's File Manager, called DEMOS 2.2 (Disk EMulation Operating System version 2.2) is obtained by procedures described above.

2.2.1 **FILE MANAGER SCREEN**

Following power up the File Manager screen looks like this: (Some versions may vary. This is a typical example)

```
          29 NOV 1983 09:50
** DEMOS VER 2.2 9.A**
BAS    .sys  COMS   .sys  CLCK   .sys
TERM   .sys  EDIT   .sys


DIR  STAT BAS  COMS CLCK TERM EDIT KEYS
```

NOTE: Only system files (.SYS extension) are displayed. Further files can be displayed with the 'DIR' command. The time display does not update to conserve power.

The function keys are programmed as follows:

1. DIR (File directory)
2. STAT (File status)
3. BAS (Basic interpreter)
4. COMS (Initialise communications)
5. CLCK (Initialise clock)
6. TERM (Terminal emulation)
7. EDIT (Text Editor)
8. KEYS (Alternate function keys)

The function 'KEYS' displays an alternative menu. See below.

DIR STAT INP SAVE TYPE REN SEND KEYS

Functions can be selected by either the appropriate function key or typing the longhand version.

HUNTER SYSTEM FILES

2.3 HUNTER's system files are not true files in the CP/M sense. They are built-in utility programs designed to support use of the computer. The function of each is described below.

2.3.1 **FILE DIRECTORY**

Displays a list of filenames. The use of wildcards is supported. See section 3.4.3, WILDCARDS.

2.3.2 **FILE STATUS**

This function displays the file status, read/write or read only, file size and remaining file space. See section 3.4.3.16, STAT.

2.3.3 **BASIC INTERPRETER**

Access to HUNTER's Basic interpreter is provided with this option. See Part 4, BASIC PROGRAMMING.

2.3.4 **INITIALISE COMMUNICATIONS**

The parameters of HUNTER's industry standard RS-232/V24 interface may be changed for compatibility with additional external equipment. See Part 6, COMMUNICATIONS.

2.3.5 **INITIALISE CLOCK**

The date and time of HUNTER's internal calendar clock may be set with this function. Calendar maintenance is completely automatic once the correct date and time have been entered. See section 3.4.3.2, CLCK.

2.3.6 **TERMINAL EMULATION**

HUNTER emulates a simple CRT terminal. The communications baud rate and protocols are first initialised with 'INITIALISE COMMUNICATIONS' function. See section 6.8, TERMINAL EMULATION.

2.3.7 **EDIT**

This activates the text editor. See section 7.

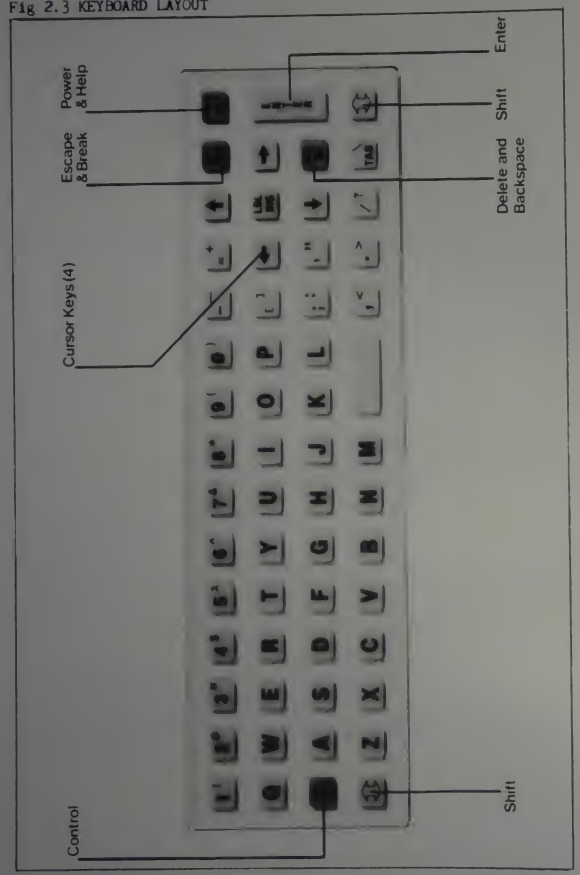
2.3.8 **KEYS**

Operation of the function 'KEYS' displays the alternate function key display line, as follows:

DIR STAT INP SAVE TYPE REN SEND KEYS

Further operation of the function 'KEYS' will return HUNTER to the original function key display line.

Fig 2.3 KEYBOARD LAYOUT



KEYBOARD

2.4 HUNTER's keyboard comprises 58 keys. Most are laid out in the standard QWERTY positions, but there are a few non-standard keys such as PWR, ESC and the cursor keys.

When held down, the keys will auto-repeat at about 8 characters per second.

2.4.1 **POWER AND HELP**

Used for switching HUNTER on and off.

NOTE: This key is generally used to ask for power, in which case it operates just for power on/off. If it has been programmed for HELP, it is necessary to press 'Shift Power' to turn HUNTER off.

2.4.2 **ENTER**

The ENTER key is used for terminating input from the keyboard. For certain inputs, replying to menus etc., the ENTER key is not used as HUNTER will respond automatically to keyboard input.

NOTE: 'ENTER' is often called 'RETURN' in user programs. The functions are identical.

2.4.3 **SHIFT**

This key accesses the legends on the upper half of the keys and must be used in combination with another key. The keys A-Z produce upper case with the shift key.

2.4.4 **CURSOR KEYS**

The cursor control keys perform a number of functions including adjustment of viewing angle, moving the virtual screen window as well as moving the cursor.

2.4.5 **DELETE, BACKSPACE AND BACKARROW**

Delete, backspace and backarrow keys within DEMOS and Basic all perform the same function, i.e. deletes the character to the left of the cursor. Some user programs in CP/M may use backspace differently by not deleting the character, but only moving the cursor back.

2.4.6 **CONTROL**

This key is similar to 'shift' in that it generates a new set of ASCII codes.

By holding down 'CTL' and pressing an alphabetic key, a control code is generated e.g. 'CTL A' clears the screen and 'CTL G' sounds the bleeper. (Note: this function is inhibited in the file manager).

Another use of control is for adjustment of the LCD display viewing angle. By pressing 'CTL' and either up or down arrow the viewing angle may be changed to suit the user. This function is always active.

2.4.7 **FUNCTION KEYS**

The CTL/FN key may also be used to generate key sequences known as function keys. These strings may be used for frequently used commands and are present within system programs (DEMOS, Basic, etc.) They are operated in a similar fashion to control, but pressing a numeric key 1-8, giving the eight function keys.

The commands implemented are displayed on the bottom line of display (this may be turned off, see the LBL key).

Within Basic programs the key may be initialised to new functions, see section 4.5.4 FUNCTION KEYS.

2.4.8 **ESCAPE AND BREAK**

'ESC' is an upper-shifted key used to break into and terminate execution of HUNTER Basic.

'BREAK' (BRK) is used to interrupt a Basic program. See 'ON BREAK' in Part 5, BASIC FUNCTIONS. Break is also available for CP/M programs and in terminal mode.

2.4.9 **INS AND LBL**

'INS' will return the virtual screen window to its original position.

'control INS' will reverse the operation of the 'SHIFT' key on the characters A - Z, providing a 'CAPS' function.

'LBL' switches the function line display on and off. LBL is an upper shifted key.

2.4.10 HUNTER's keyboard is built for extended use and long life. The keyboard will withstand assaults from water, coffee and mechanical shock, but it can be damaged by impact with sharp objects.

2.4.11 **KEYBOARD RE-DEFINITION**

Keyboard keys are each equated to the characters they represent by unique numerical codes, as listed in section 9.2, ASCII CHARACTER SET. These values are defined according to the American Standard Code for Information Interchange (ASCII).

The ASCII definitions of each key for the keyboard are contained in 116 consecutive RAM locations named KEYBUF. The upper shift keys are defined in the first 56 locations, followed by the 56 lower shift keys. These memory locations form a 'map' of the physical keyboard. On power up, the HUNTER automatically defines these locations to a standard configuration. However, the programmer has the option to be able to define special keyboard arrangements.

The schematic of the keyboard, see section 9.6 KEYBOARD MEMORY MAP, shows the value in KEYBUF for each key.

It is important to note that on powering up HUNTER, the keyboard will revert to the standard version. The programmer should define the special keyboard requirements at an early stage in the program and ensure that the re-definition occurs each time the program is run.

2.4.12 **SPECIAL CODES**

Special codes are assigned to keys whose functions are:

(1) Control key
(2) Shift key (latched)
(3) Help key
(4) Momentary shift key
(5) Power On/Off key
(6) Virtual screen window up key
(7) Virtual screen window down key
(8) Virtual screen window right key
(9) Virtual screen window left key
(10) Virtual screen right
(11) Virtual screen left
(12) Label key
(13) Insert key

In order for the operating system to detect a change of these keys, the SPELFLG location must be cleared. Changes modifying the special codes should only be done under software control.

NOTE: 'Enter' is not a special code: it is simply CR (Decimal 13) !

LATCHED SHIFT CODE 129 (81H)
This key should be defined in both shift halves in order to function correctly! This function gives a 'toggle' action, changing shift with each operation.

MOMENTARY SHIFT CODE 130 (82H)
As HUNTER powers up in the lower shift, it is important for the momentary shift code to be defined in the lower shift half of KEYBUF. This function gives a 'conventional' shift requiring simultaneous depression of shift and the desired key.

CONTROL CODE 132 (84H)
On detection of this key the control code for the key pressed is derived from the code located in the upper shift half of KEYBUF, if this key has a valid 'control' equivalent.

CONTROL CODE 133 (85H)
On detection of this key the control code for the key pressed is derived from the code located in the lower shift half of KEYBUF, if this key has a valid 'control' equivalent.

HELP CODE 134 (86H)
The detection of this key causes the firmware to enter the HELP text mode. Help mode is exited by pressing the 'HELP' key again.

ON/OFF CODE 136 (88H)
The power on/off key may be relocated to any position on the keyboard. After HUNTER has been powered off the power key defaults back to the top right hand position of the keyboard.

VIRTUAL SCREEN WINDOW UP CODE 138 (8AH)
VIRTUAL SCREEN WINDOW DOWN CODE 139 (8BH)
VIRTUAL SCREEN WINDOW RIGHT CODE 140 (8CH)
VIRTUAL SCREEN WINDOW LEFT CODE 141 (8DH)

The above keys allow the user to view the contents of the virtual screen. The keys are completely transparent to the user program.

LABEL CODE 143 (8FH)
Switches the function key display line on and off. The key is

transparent to the user program.

INSERT CODE 144 (90H)
Returns the virtual screen window to its original position after being moved by the window control keys. The key is transparent to the user program.

2.4.13 ON/OFF KEY

Unlike most computers, HUNTER's ON/OFF switch is not actually in control of power removal. The key is seen by the software scanning the keyboard and the actual decision to turn off is up to the software itself.

If a machine code program places HUNTER into an endless loop, it will be impossible to turn off the machine. This may be overcome by removing the main batteries.

SCREEN

2.5.1 HUNTER has a large LCD (Liquid Crystal Display) screen. It can present up to 320 characters on 8 lines of 40 characters each, just like a small video terminal.

The character set includes upper and lower case alphabets and special characters. Additionally, a graphics display of 240 x 64 pixels is supported.

HUNTER's display is designed for use in bright sunlight where other displays (the red 'LED' or green 'vacuum flourescent', for example) become invisible.

Because it works by contrast, rather than emitting its own light, it is perfectly visible no matter how bright the light is. But a word of caution: try to avoid leaving HUNTER exposed to direct sunlight for prolonged periods - the delicate chemicals in the LCD can be damaged.

The 'Cursor' shows where data entered on the keyboard will appear.

The screen is protected by an acrylic window, as strong as glass. Like glass, it can be scratched, so please take care. The window should only be cleaned with chamois or lens cleaning cloth.

HUNTER is built to last - please take care of it.

2.5.2 SCREEN MODES

The screen can be used in two ways:

: Text mode
: Graphics mode

Text mode provides only character support with the standard 8 lines of 40 characters displayed. Internally, there is a "virtual screen", see section 2.6, supporting 24 lines of 80 characters. This mode can support any standard CP/M program, e.g. SuperCalc or dBase II. It can be used in this fashion due to the speed at which the LCD screen can be refreshed in this mode. This is the only mode supported for CP/M based programs.

Graphics mode will support characters of five different sizes, reverse video and full graphic capabilities. There is no "virtual screen" associated with this mode. It is only supported from Basic, see section 5.

2.5.3 **Screen Control**

The text screen will support "clear" and screen addressing. The screen is addressed throughout the virtual screen, that is the addressing range is from 0 to 79 horizontally and 0 to 23 vertically.

Cursor addressing is a three character sequence sent to the screen handler. It is of the form:

Cursor addr control, x-coordinate, y -coordinate

The cursor address conrol is 0FH (15 dec).

The x coordinate is a number 0 - 79.

The y coordinate is a number 0 - 23.

The screen may be cleared by sending a control A or 01H character to the screen handler.

VIRTUAL SCREEN

2.6 The eight by forty characters on the LCD are only one sixth of the number of characters contained within the virtual screen. The LCD acts as a window onto the virtual screen. The contents of the virtual screen may be inspected at will by using the cursor control keys in their shifted state.

2.6.1 SCREEN SIZE

HUNTER's virtual screen is 80 characters by 24 lines. The virtual screen behaves very much like a terminal with automatic scrolling. When the cursor reaches the bottom line, the rest scrolls up losing the top line.

2.6.2 MOVING THE WINDOW

The window may be moved over the virtual screen either manually by the user, or automatically by the virtual screen handler resident in the operating system and as such is transparent to the user.

2.6.2.1 Manual Movement

1) Shifted state cursor control keys (looking keys):

By using the cursor control keys in their shifted state, the window can be made to move in the direction marked on the key, i.e. press the 'shift' key, hold it down, and then press the desired cursor control key. When this operation is performed it will be noticed that the cursor on the LCD (a blank flashing block) will change its position on the LCD and quite often disappear. This will happen because the cursor position refers to the position in the virtual screen, and when the window is moved the cursor position relative to the window position in the virtual screen will change.

2) Function "cursor control keys"

By pressing CTL/FN and either horizontal arrow key, then either the left or right 40 characters of the screen can be seen. This function is useful for easy reading of wide screen formats.

3) 'INS' Key:

Pressing the INS key will return the window to its original position prior to use of the looking keys, as outlined in (1) above.

4) Transparent Movement:

The transparent movement of the virtual screen window is primarily controlled by inputs, both from the keyboard and the serial port. If HUNTER is required to accept an input from, say, the keyboard then the window will automatically be moved to allow the viewing of the immediate area of the virtual screen centred around the present cursor position, thus displaying any prompts which may have been required by either a Basic program or a CP/M type program.

It must be noted that printing text onto the virtual screen will not move the window. However, if any part of the virtual screen which is presently in view on the LCD is changed by, say, a PRINT statement, then these changes will be reflected on the LCD, thus always giving a true representation of the virtual screen.

If HUNTER is accepting input from the keyboard and the cursor reaches the forty-first position of the virtual screen, then the window will be moved laterally so that the cursor is now positioned mid-way across the LCD, as soon as the forty-first character is entered, thus displaying the previous 20 virtual screen characters and the following 20 virtual screen characters.

If HUNTER is waiting for input and the looking keys are used to move the window, a valid input will cause the window to automatically revert to its original position prior to the use of the looking keys. This facility does not force the user to press the 'INS' key in order to restore the original position of the window when in this mode.

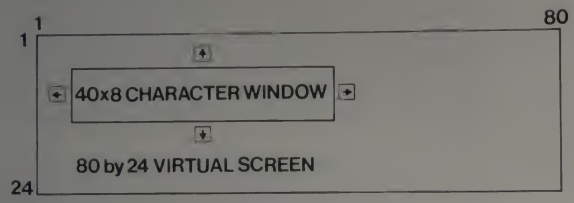
For further information on cursor addressing see LOCATE in Part 5, BASIC FUNCTIONS and section 2.5.3.

NOTE: The virtual screen window does not operate in Graphics mode.

2.6.3 MOVING THE CURSOR

To move the cursor in the required direction it is only necessary to press the appropriate cursor control key.

Fig 2.4 VIRTUAL SCREEN

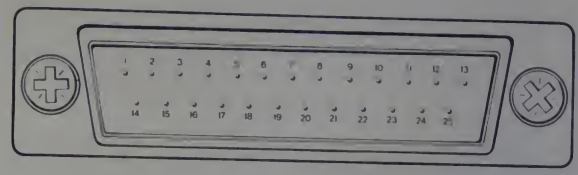


note The shifted cursor control keys ↑ ↓ ← → move the window over the virtual screen in the direction of the arrows.

2.7 COMMUNICATION

HUNTER has an industry-standard RS-232/V24 serial communications port, and will talk directly to most computer peripherals, modems and other devices.

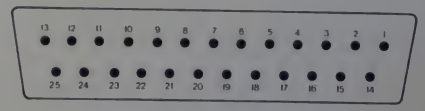
Fig 2.5 MALE (HUNTER)



The port communicates via a standard male 25-way 'D' type connector. The pin connections and other data are detailed in Part 6, COMMUNICATIONS.

NOTE: Never, never try to force a mating connector home. Check that the connector is the right way up (like this):

Fig 2.6 FEMALE (CABLE)



If the connector won't fit easily, it probably won't fit at all.

FIND OUT WHY

HUNTER's communication format is usually set up by its applications program for specific situations, and does not require operator attention.

Sometimes, a user program will present a 'menu' selection for operator choice. In this event, selection of the desired communications mode will automatically set the communications parameters without further action.

Communications parameters can also be set manually. An internal program, 'COMMUNICATIONS PARAMETERS', can be accessed or called by a user application program. See Part 6 COMMUNICATION, for details.

Fig 2.7 BATTERY INSTALLATION

BATTERIES

2.8 HUNTER is battery powered. It runs for a long time on a set of standard 'AA' size cells obtainable anywhere.

A second rechargeable battery is permanently built into HUNTER so data is not lost even if the main batteries are removed. This battery should never need to be replaced.

2.8.1 BATTERY INSTALLATION

The batteries are spring-loaded to make contact and retained by a threaded plug. When installing batteries, follow this procedure:-

1. ONLY USE A COIN in the battery plug. Screwdrivers, etc., will damage the slot and HUNTER's appearance.
2. Feed in the cells, **positive inwards**. Don't drop the 'AA' cells in vertically - they can be damaged. Instead, slope HUNTER slightly.
3. Take the battery plug and, with finger pressure only, press into the battery compartment and turn.
4. Only when the thread is started, use a coin to screw the plug home.

The battery plug should stand about 2mm proud from the case wall.

NOTE: Clockwise inserts the plug
Anti-clockwise removes the plug

2.8.2 MAIN BATTERY - PRIMARY

We strongly recommend the use of Alkaline-Manganese batteries similiar to Mallory MN1500.

We do not recommend the use of cheaper zinc-carbon cells for routine operation, since these can suffer electrolyte leakage, have shorter lives and can introduce technical problems.

Alkaline cells will produce approximately 45 hours of HUNTER use when running a typical application program, and almost indefinite storage life. There is no need to remove cells from HUNTER during storage or shipment but remember that capacity will reduce with time.

2.8.3 BATTERY - RECHARGEABLE

Only Nickel-Cadmium (Ni-Cd) cells are to be used. Under no circumstances must any other type of rechargeable cell be installed. We recommend Berec type NCC50 cells of 500 mAH capacity.

2.8.4 Low Battery Warning

HUNTER is designed to provide adequate warning of impending battery failure and will protect user's data, even in the worst situations. This warning facility has been improved in Hunter operating systems Ver.9G onwards.

When HUNTER's batteries become low, HUNTER will give warnings to the user. Since it is usually essential that all data and programs are secured, it is recommended that HUNTER is switched off as soon as practicable and the batteries either replaced with a fresh set or, if rechargeables are fitted, put on charge.

Low Battery warnings are repeated as follows:

a) every 5 seconds if the HUNTER is switched on, but quiescent,

b) after every keyboard key depression

or,

c) every 0.5 seconds if characters are being received on the RS232 port.

If ten consequentive warnings are ignored, HUNTER automatically switches off. When this occurs, HUNTER is placed automatically in a CONT mode (see section 3.4.3.4). This means that when power is restored, program execution will continue from precisely where it was prior to the interruption.

HUNTER has two screen modes, text and graphic (see section 2.5). Battery low warning is slightly different in the two modes:

a) **Text Mode**

When a low battery is detected, the top line of the screen displays:

* WARNING BATTERIES ARE LOW *

HUNTER bleeps twice and then the original text is restored until the next warning.

The text screen contents are preserved and not affected by the warnings. Following power restoration, the screen is restored to its pre-warning state.

b) **Graphics Mode**

When a low battery is detected, the top line of the screen displays:

* WARNING BATTERIES ARE LOW *

in reverse video and using the smallest character set (CHARO). This message stays on the screen between battery warnings. HUNTER also bleeps twice. The warning message destructively overwrites the top line of the screen, the contents of which are lost and must be restored by the user's program.

If an attempt is made to use HUNTER without replacing or recharging the batteries, Battery Low warning is displayed and HUNTER immediately turns off again. It is worth commenting that flat batteries can partially recover after being switched off such that operation could continue for a short period. It is not recommended that this should be done in practise, particularly with rechargeable batteries which could then become totally exhausted. In the case of Ni-Cd batteries, this can lead to permanent damage to the cells.

2.8.5 POWER SAVE

An automatic power conservation system has been incorporated into HUNTER, enabling the power to be removed from circuits when they are not in use.

Usually, the computer will enter power save mode whenever it is awaiting a keyboard entry. However, a few programs defeat this facility by not using standard CP/M console calls and have a correspondingly higher overall power consumption. Please consult Husky Computers for advice in specific cases.

2.8.6 BATTERY CHARGING

HUNTER is optionally available with rechargeable cells and a mains (line) powered charger.

This arrangement allows cells to be recharged in HUNTER, and for alkaline cells to be quickly substituted if the user forgets to recharge!

But BE CAREFUL - **NEVER** connect the charger when alkaline batteries

are installed. They will NOT re-charge. Instead, they may explode, jam in the battery tube or leak corrosive chemicals.

See section, 8.6, BATTERY CHARGER, for further information regarding the use of HUNTER's battery charger.

2.8.7 CONTINUOUS CONNECTION

HUNTER can be powered permanently from the charger by simply leaving the unit connected. In this mode, the rechargeable cells will be kept 'topped up' by the charger despite the continuous current drain. Of course, a fully discharged battery will take longer to recover than if HUNTER is powered down.

2.8.8 Auto Timeout

To conserve battery life, HUNTER will automatically switch off after a period of inactivity. A warning bleep is emitted every 2.1/2 minutes to remind the user that HUNTER is turned on. This timeout does not operate in Terminal Emulation (see section 6.8). The timeout value can be modified using POWER in Basic (see section 5.17.7) and has a default value of 5 minutes.

This page intentionally left blank.

2.9.1 PANIC

HUNTER is designed, built and quality tested for robustness in every sense including robustness of program execution.

HUNTER's microprocessor system has wide design tolerances (much more than conventional computers) to guarantee absolutely reliable execution of millions upon millions of machine-code instructions every hour. Its physical construction protects it against mechanical disturbance, while the conventional computer's Achilles heel - external electrical interference - is virtually eliminated.

Because of these factors, HUNTER is most unlikely ever to mis-execute a user's program or behave in a way that is not thoroughly predictable, given sufficient insight.

2.9.2 CRASHES

However, there are some specific situations that can cause HUNTER to mis-execute its internal programming, or "Crash".

The commonest causes are:

2.9.2.1 Illegal System Calls

System calls from Basic are not 'trapped' (this would restrict user programming) and can cause mis-execution if not valid.

2.9.2.2 Invalid user machine code

User assembly-level programs or subroutines can easily cause mis-execution by containing, for instance, invalid jump instructions.

2.9.2.3 Physical degradation

Ingress of water, corrosion of internal parts, damage to components through excessive shock or persistent high level vibration can reduce electronic tolerances.

2.9.2.4 Operation outside specified temperature range

HUNTER is specified for operation in the range 0 to 55°C. Operation above 55°C reduces tolerances, while below 0°C battery capacity becomes severely restricted. Sub-zero temperatures also slow LCD screen response time substantially. HUNTER is unlikely to mis-execute program simply because of low temperatures, however, in the range 0°C to -20°C.

2.9.3 **SYMPTOMS**

"Crashes" are identified by these symptoms:

2.9.3.1 **Keyboard Lock-out**

HUNTER's keyboard is entirely "soft" to allow user re-definition of key functions, **including** the power key. If HUNTER's internal program is caused to 'Crash', power control may be lost, HUNTER will not switch off.

2.9.3.2 **Clock wipe-out**

Following a mis-execution episode, HUNTER's calendar clock registers may be corrupted.

2.9.3.3 **Program Corruption**

The most serious consequence of mis-execution is when the micro processor runs 'wild', writing data randomly to all parts of memory and, occasionally, corrupting user programs. This failure is generally catastrophic, and is unlikely to go undetected. The most likely consequence is that any attempt to 'RUN' the corrupted program will result in keyboard lock-out (see above). Attempts to 'LIST' corrupted programs may also result in lock-out.

2.9.3.4 **I/O Corruption**

I/O selections may be affected by mis-executions in a random fashion. In some cases spurious options may appear in the parameter selection screens after a 'Crash'. In this event, hold down one of the cursor control keys until recognisable options appear.

2.9.4 **RECOVERY**

2.9.4.1 **Program Corruption**

If a mis-execution has occurred, any user program stored in RAM memory must be viewed with suspicion. The safest solution, assuming that the use of keyboard is still available, is to type 'NEW', followed by re-loading of the program.

Remember that even 'LIST' may result in keyboard lockout, although holding the power key down will generally restore control eventually.

Most types of corruption (corrupted lines, spurious line numbers) are unrecoverable in Basic and can only be eliminated by 'NEW'.

2.9.4.2 **Keyboard Lock-out**

If keyboard lock-out (failure to power down) occurs, it is necessary to power down HUNTER by removing the battery cap.

HUNTER should power-up again normally. If desired, try running the user program, but be ready to clear any lock-out that occurs using the above methods. If lock-out persists, type 'NEW' after restoring operation.

2.9.4.3 **System Lock-out**

In very rare cases, HUNTER may fail to restart correctly after a mis-execution episode. This is possible in a program that uses auto-start, see section 4.12, AUTO POWER FEATURE, and contains an invalid system call.

Any program corruption resulting could leave the auto-start flag set, but cause a lock-out when power-up is attempted.

The solution is to apply the 'ESC' sequence immediately after power-up, in order to reset the auto-start flag.

2.9.4.4 **File corruption**

Because files are not stored in Hunter's workspace (RO) area, mis-execution episodes are much less likely to cause corruption of this part of memory. In any case, the checksum facility will indicate if damage has ocurred. Current experience suggests that storing a 'back up' file in a different memory page virtually guarantees data integrity against the worst possible disasters.

A checksum error can result in a "no file" message being displayed in response to an attempt to use the file, even if the directory entry is present. In this event, the only recourse is to re-load the file.

2.10 BATTERY LIFE

2.10.1 HUNTER's power control system is designed to optimise battery life by adapting power consumption to the computer's level of activity. This means that power consumption varies according to the level and nature of use, making calculations of battery life slightly more complex than usual. However, if it is known how long HUNTER will be used in its different modes, it is possible to calculate with reasonable accuracy the expected life of a set of batteries.

2.10.2 Battery capacity is measured in mAH (milliamp hours) and by knowing the battery type in use, life may be calculated.

Battery capacity is affected by a number of factors including age, charging method, ambient temperature, etc. However, the capacity typically available from the two types of battery used by HUNTER is as follows:

a) DISPOSABLE Duracell type MN1500 (I.E.C. LR6) capacity 900 mAH.

b) RECHARGEABLE Berec type NCC50 (I.E.C. KR 15/51) capacity 425 mAH.

Because capacity can vary significantly, the application planning should always assume reduced available capacity. There are no firm guidelines for this, but 50% of capacity is considered conservative.

A factor affecting apparent battery life is the "end point", or voltage at which the battery is considered to be exhausted. Hunter sets this end point at 4.6 - 4.8 volts for battery warning.

This value yields virtually 100% of Nickel Cadmium batteries capacity and gives good results for Alkaline cells, but means that manufacturer's specifications for capacity may some times be misleading.

2.10.3 HUNTER's power consumption has three principal modes:

a) POWER SAVE 15mA
(awaiting keyboard or serial port entry)

b) OPERATING 90mA (208K)
(running a program for calculation, data storage, screen updating or any operation involving continuous execution of user program).

c) COMMUNICATING
(connected to a fully equipped RS232 port and operating) 125mA (208K) 115mA (80K)

As can be seen, the POWER SAVE current is very low. Although the screen is on and the keyboard responsive to any key depression, this is the condition in which HUNTER is expected to be most of the time.

A typical application example might be a salesman using HUNTER for order entry purposes at a number of customer's premises and returning to base for communication.

2.10.4 **Example of Power Consumption**

The day is 7.5 hours long, during which he makes five calls. At each call he uses HUNTER for 40 minutes for stock checking and order entry purposes. The communication time takes 15 minutes, with a further 5 minutes left powered up.

Of the 40 minutes use at each call, only 5 minutes is considered 'operation' since most of the time HUNTER is waiting for keyboard entry.

Each Visit

Operating Power $\frac{80 \times 5}{60}$ = 6.67 mAH

Power Save Power $\frac{15 \times 35}{60}$ = 8.75 mAH

Total Visit Power = 15.4 mAH

Total Day's Power:

5 visits at 15.4mHA = 77 mAH

Communicating $\frac{115 \times 15}{60}$ = 28.75 mAH

Waiting after coms $\frac{40 \times 5}{60}$ = 3.33 mAH

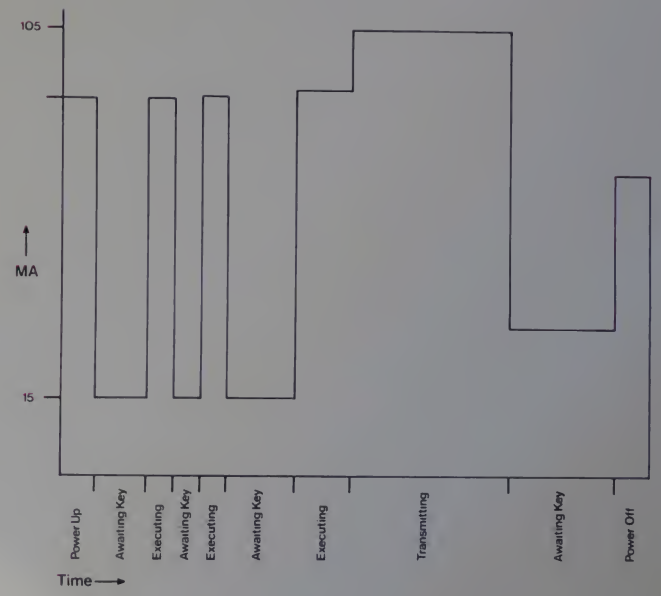
Day Total = 109 mAH

Total Time in use through the day: 3 hours 40 minutes.

It would be expected that the HUNTER would last approximately 4 days (or 14 hours 40 minutes total) on a set of rechargeable cells. Alternatively, HUNTER would last 10 days (or 36 hours 40 minutes) on a set of manganse alkaline cells.

BATTERY LIFE

A chart of the current used by a HUNTER may be shown as follows:



This illustrates a period of use of the HUNTER. In practise, the periods of power save are much longer than that illustrated.

FILE MANAGER

CONTENTS

DEMOS — FILE MANAGER

3.1 HUNTER's file manager DEMOS (Disc EMulation Operating System) manipulates files and runs CP/M programs. A large range of commercially available CP/M software will run on HUNTER with no modification

After switching HUNTER on, the 'Welcome Message' is displayed and the following screen can be obtained by pressing any key:

```
        20 Jan  1984  13:04:27
     **DEMOS VER 2.2 9.A**
     BAS   .sys  COMS  .sys  CLCK  .sys
     TERM  .sys  EDIT  .sys


DIR  STAT  BAS  COMS  CLCK  TERM  EDIT  KEYS
```

The bottom line shows the titles of the soft keys. See section 2.2, FILE MANAGER DISPLAY.

The File Manager has several important functions:

(1) All programs such as word processors, spreadsheets and the Basic Interpreter are run from the File Manager.

(2) Various system files can be selected, for example, setting up the communication protocols.

(3) Information on the size of the files present (STAT) or file operations like transmitting a file (SEND) can be performed.

3.1.1 FILE NAMES

Files are named following the general format of CP/M. They consist of up to eight alphanumeric characters separated by a period from an extension of three alphanumeric characters. The extension informs the operating system of the type of file, if the user so wishes. Generally, the extensions follow these conventions:

.HBA - HUNTER Basic file containing program source code.

.COM - A CP/M compatible object code file which can be loaded and run independently of HUNTER Basic.

.DTA - HUNTER Basic data file.

.TXT - Any ASCII text file.

.SYS - A 'psuedo file', generally in ROM.

3.1.2 KEYBOARD COMMANDS

By use of the control key together with another key the editing functions, see TABLE 3.1 KEYBOARD COMMANDS, can be implemented under the File Manager:

TABLE 3.1 KEYBOARD COMMANDS

'control C' - Reboots when at the beginning of the line.

'control E' - Causes physical end of line.

'control H' - Backspace one character position.

'control J' - (Line feed) terminates input line.

'control M' - (Return) terminates input line.

'control R' - Retypes the current line after new line.

'control U' - Removes current line after new line.

'control X' - Backspace to beginning of current line.

PROGRAM EXECUTION

3.2 CP/M, and HUNTER Basic programs may be run from the File Manager. In both cases the program is loaded into the working RAM page and executed either under the Basic interpreter or directly. CP/M and Basic program files are stored together in RAM file space, and are treated as if they were stored on disk.

3.2.1 BASIC PROGRAMS

HUNTER Basic programs can be run without explicitly invoking the interpreter. Entering a filename which has the .HBA extension will cause the Basic program to be loaded and the interpreter to be executed. For example, if the directory reads as follows:

```
>DIR
PROG1  .HBA  WORDS  .COM  PROG2  .HBA
NAMES  .TXT
```

Then typing:

```
>PROG1.HBA
```

executes the HUNTER Basic program PROG1.HBA. The file extension .HBA must be specified. To enter the Basic interpreter use the File Manager command 'BAS', function key 3. See Part 4, BASIC PROGRAMMING, for further information on HUNTER Basic.

3.2.2 CP/M PROGRAMS

Object code files with .COM extensions may be executed in the standard CP/M manner. The filename entered, for example using the directory above:

```
>WORDS[.COM]
```

will cause the program to load and run. The .COM file extension is optional.

FILE STRUCTURE AND MEMORY ORGANISATION

3.3.1 HUNTER is capable of storing a great deal of differing information within its large memory. There may be Basic programs, text files, data files or object code CP/M programs.

Superficially the operating system handles its memory in a similar manner to disk backing stores, and indeed may be considered as a 'RAM Disk'. However HUNTER is much faster than any disk system and incorporates special facilities to enable the memory to be efficiently utilised.

HUNTER's File Manager DEMOS (Disk EMulation Operating System) manipulates files and runs CP/M programs. The File Manager environment is totally unconnected with Basic.

3.3.2 **ENVIRONMENT AND STRUCTURE**

The File Manager facility is located in firmware, so there is no overwriting of existing programs in execution RAM. A CP/M call 0, or a JMP 0000 both cause a "warm" start which executes the File Manager in ROM. This would be a typical method of ending a .COM program. Entering the File Manager on power up is equivalent to a "cold" start, in which the operating system is completely reset, although programs in R0 are preserved.

HUNTER's File Manager will implement several commands and load and execute .COM files. Files may be erased, renamed or saved from memory, ASCII files may be displayed on the screen, as can the file directory. Finally, there is a facility for loading and dumping the files via the serial port in either Intel hex format or 8 bit binary format.

The File Manager is structured around the CP/M compatible operating system and file structure. The RAM filespace, organised in exactly the same manner as disk based files, can be accessed through the system calls. Utility commands can be quickly selected from the soft keys.

3.3.3 **FILE SPACE**

Files are stored, as on magnetic disk, in records of 128 bytes, grouped by 16's into 2K bytes. This means that the smallest file size is 2K bytes, but, as with magnetic media, allows for very large files to be stored.

The memory space is allocated by the file manager and is transparent to the user.

3.3.4 **FILE ERRORS**

Each time a file is modified an error checksum is calculated and stored in the directory. This operation is completely transparent, as is each accuracy check when the file is accessed for reading or executing. If any discrepancy is found then a "no file" message is displayed.

3.3.5 **HUNTER MEMORY ORGANISATION**

HUNTER is designed for very large RAM memories and 48k ROM, see Fig 3.1, HUNTER MEMORY MAP and Fig 3.2, HUNTER FILE ACCESS. The memory is organised into pages: one page of ROM operating system, one page of RAM execution memory and further pages of RAM file space. All pages except for R0, the execution memory (also known as the Transient Program Area, or TPA) are limited to 48k. R0 extends to 54k, although the top 6k together with the 10k of common RAM are never paged out. There is, therefore, up to 54k of RAM for programming.

Common RAM, see Fig 3.3, SYSTEM RAM MEMORY MAP, is used to link together the various pages of RAM and ROM. It organises the paging and contains the jump tables for CP/M compatibility. The operation of paging is totally transparent to the user so HUNTER is seen to have 54k program space with a powerful 48k operating system and Basic interpreter in ROM, and a block of file space.

3.3.6 **BASIC PROGRAMMING**

Since the Basic interpreter resides in ROM, the whole 54k of R0 is available for a Basic source program, variables and arrays. Simple variables are stored with the source program and Basic may access files for further storage.

3.3.7 **CP/M PROGRAMMING**

Object code programs must be executed from R0. Again, there is 54k RAM available for the program, plus the file space. Normally, the program will reside in the HUNTER as a .COM file, and will be loaded and run by the File Manager. However, the File Manager is very versatile, and provides flexibility in file manipulation and execution.

3.3.8 FILE ORGANISATION

The pages reserved for file storage have, as magnetic media, a "sector allocation map" and the actual directory, split into 16K extents. Any good text on CP/M file storage will give an adequate explanation, which is also true for HUNTER.

This means that 2K byte blocks of a file are in no particular order within the filespace RAM, making large files and fast updating possible.

Fig 3.1 HUNTER MEMORY MAP

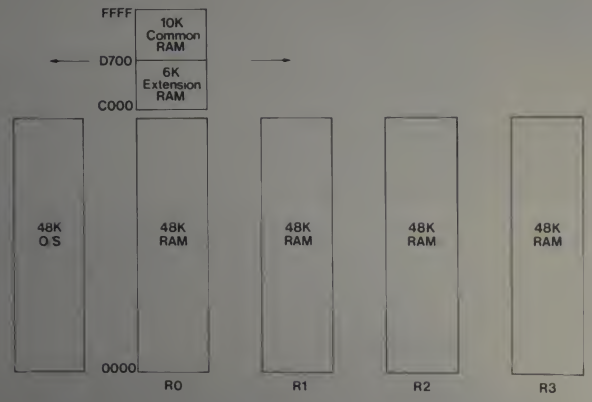


Fig 3.2 HUNTER FILE ACCESS

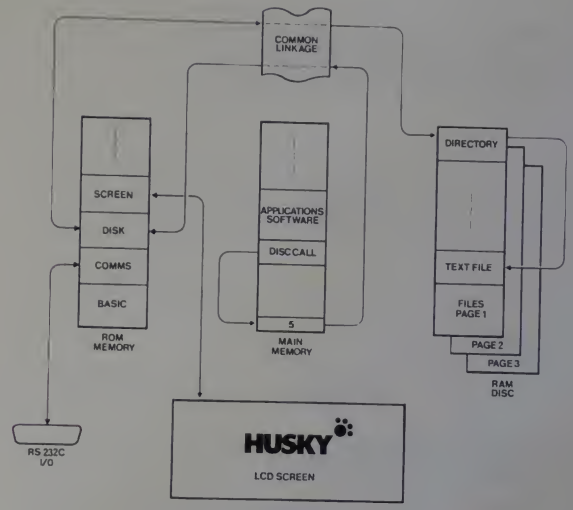
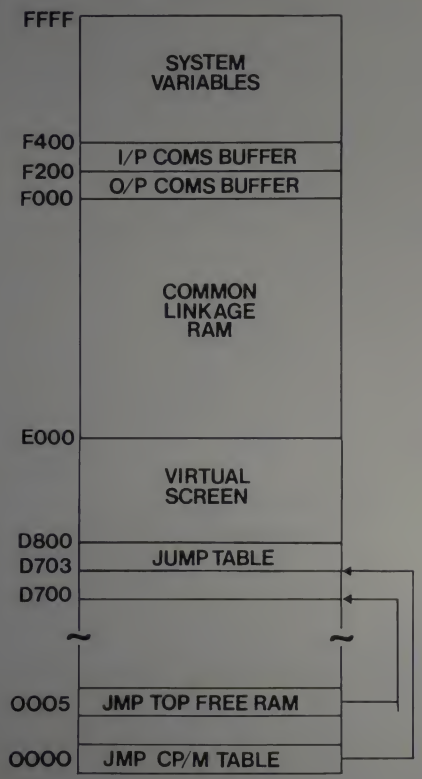


Fig 3.3 SYSTEM RAM MEMORY MAP



FILE MANAGER COMMANDS

3.4.1 PROMPT

The prompt character, which is displayed when the File Manager is ready to receive a command, is the greater than sign '>'. If this is not displayed, then no input is expected.

3.4.2 COMMAND SYNTAX

Commands are typed in with their arguments (usually filenames) separated by spaces. Wildcard characters may be specified. Commands may be in upper or lower case alphabetic characters.

3.4.3 WILDCARDS

DEMOS supports the use of wildcards '?' and '*'. The '?' character will allow DEMOS to search for file names and disregard the characters replaced with '?' For example, suppose, there are five files with the names FILE1.TXT - FILE5.TXT. The following DIR command would only list those files:

```
>DIR FILE?.TXT
FILE1.TXT   FILE2.TXT  FILE3.TXT
FILE4.TXT   FILE5.TXT
```

or

```
>DIR A?FILE.COM
AAFILE.COM  ACFILE.COM  A4FILE.COM
```

The '*' wildcard replaces the entire extension or file name. The example below will list all files with the extension .HBA:

```
>DIR *.HBA
PROG1.HBA  PROG2.HBA  START.HBA
```

or

```
>DIR MYFILE.*
MYFILE.TXT  MYFILE.COM  MYFILE.HBA
```

To list all the files held in HUNTER the command 'DIR' is used.

A combination of wildcards can also be used:

```
>DIR ??FILE.*
MYFILE.TXT  MYFILE.COM  MYFILE.HBA
AAFILE.COM  ACFILE.COM  A4FILE.COM
MRFILE.HBA
```

Wildcards may be used with other File Manager commands apart from DIR, for instance, as follows with the ERA command:

```
>ERA BIG*.*
```

this will erase any file starting with 'BIG' and any extension for example, BIGER.FIL, BIG.HBA or BIGGLES.COM would all be deleted.

Note that, as a precaution, Hunter will query the erasure of each file in turn, requiring operator confirmation at each stage.

BAS

3.4.3.1 **COMMAND** : BAS

EXPLANATION : Execute HUNTER's Basic interpreter.

EXAMPLES : When the Basic interpreter has been entered the following screen is displayed:

```
┌──────────────────────────────────────┐
¦Hunter Basic Interpreter              ¦
¦READY                                 ¦
¦                                      ¦
¦                                      ¦
¦                                      ¦
¦File Run  Edit List Save Load Kill Syst¦
└──────────────────────────────────────┘
```

REMARKS : HUNTER's Basic interpreter is executed from ROM. If the last program was a CP/M program then an automatic NEW resets Basic, otherwise any Basic programs remain as before. Variables also remain unaffected. For complete information on Basic programming refer to Part 4, BASIC PROGRAMMING.

CLCK

3.4.3.2 **COMMAND** : CLCK

EXPLANATION : Set the internal calendar clock to the desired date and time.

EXAMPLES : Calendar clock initialisation screen:

```
        13 Oct 1983  11:55:00
  Time setting mode
```

REMARKS : To set the internal calendar clock proceed as follows:

a) Using the cursor right and cursor left keys, step the cursor to the item you wish to set. The seconds cannot be set and are always initialised to zero when synchronising the clock (see below).

b) Using the cursor up and cursor down keys, step the value selected until it is correct.

The months step in the following manner:-

Nov, Dec, Jan etc. ; Cursor up
Jan, Dec, Nov etc. ; Cursor down

c) Set the 'minutes' to one minute ahead of the present time.

d) Press 'Enter'. HUNTER will display:-

'Press ENTER to synchronise'

e) When the time code, pip tone, or other reference arrives press ENTER.

HUNTER's clock will run from zero seconds in exact synchronisation with the external reference.

f) The calendar automatically accounts for leap years.

g) If an attempt is made to specify an incorrect day value, e.g. 31 st April, HUNTER will display the warning:-

Error - Incorrect day value

The error must be corrected before synchronising the clock.

HUNTER's clock can be used to time-stamp entries, automatically label printouts or even initiate program execution automatically like an alarm clock.

See Part 5, BASIC FUNCTIONS, for details on TIME$, DAY$ and DATE$ functions and Section 3.5.3.40 and 3.5.3.47 for details of CP/M clock calls.

COMS

3.4.3.3 **COMMAND** : **COMS**

EXPLANATION : Initialise the communications parameters.

EXAMPLES : Two screens are used to select the communications parameters as follows:

```
Transmission Parameters

Rate-1800      Prtcl-none      Pty-none

CTS-n DTR-n    LF-n Echo-n  T/O-no NULL-0

press ENTER if acceptable
```

```
Receiving Parameters

Rate-1200      Prtcl-none      Pty-none

RTS-off  DSR-n     DCD-n  T/O-No    Serig-off

press ENTER if acceptable
```

REMARKS : See section 6.4, COMMUNICATIONS PORT SOFTWARE, for information on selecting communication parameters.

NOTE: Communication parameters are not reset when HUNTER is powered off.

In CP/M, the serial communications interface is assigned the following logical devices:

Transmit - LST:, PUN:
Receive - RDR:

CONT

3.4.3.4 **COMMAND** : **CONT** filename

EXPLANATION : Execute program with 'continue' option.

EXAMPLES : >CONT PROG1.COM

This will load and run the CP/M program 'PROG1.COM'.

The three character file type extension does not have to be specified for .COM files but must be included for HUNTER Basic files, for example, a HUNTER Basic program has the name 'SORT.HBA', the following would load and execute the program via the Basic interpreter:

>CONT SORT.HBA

REMARKS : If at any time during execution of a program initiated with 'CONT', HUNTER is switched off or power down takes place, switching HUNTER on will cause the program to continue execution from the point at which power was removed.

Escape from 'CONT' mode during program execution is made by pressing 'Control C' when power is off, and then pressing 'pwr' to power on while keeping 'Control C' depressed. Two 'beeps' should be heard, when the keys should be released. Then enter the special security number, the default for which is 56580.

If an error is made, the Hunter will 'beep' and return to program execution.

DIR

3.4.3.5 **COMMAND** : DIR [filename]

EXPLANATION : Display the file directory.

EXAMPLES : >DIR
ONEPROG.HBA TWOPROG.COM FINAL.HEX
ANOTHER.1

DIR can also be used with wildcards as follows:

>DIR FILE?.TXT
FILE1.TXT FILE2.TXT FILE3.TXT
FILE4.TXT FILE5.TXT

or

>DIR A?FILE.COM
AAFILE.COM ACFILE.COM A4FILE.COM

or

>DIR *.HBA
PROG1.HBA PROG2.HBA START.HBA

REMARKS : A combination of wildcard characters can also be used.

If the directory is too long to fit on the display screen then use of the cursor control keys can scroll back the virtual screen to see the earlier entries.

The virtual screen contains 24 lines, if a file directory extends to more than 24 lines, the top lines will scroll off the screen. Use of wildcards may help in limiting the number of directory lines displayed.

EDIT

3.4.3.6 **COMMAND** : EDIT [filename]

EXPLANATION : Edits either ASCII text or Husky Basic (.HBA) files.

EXAMPLES : >EDIT PROG1.HBA

This will edit a Husky Basic file.

>EDIT TEXT

This command will edit on ASCII file TEXT.TXT. The .TXT is defaulted.

>EDIT NEWS.DAY

Edit the ASCII file of the above name.

REMARKS : Further information on the editor can be found in section 7.

ERA

3.4.3.7 **COMMAND** : ERA filename

EXPLANATION : Erase the specified file.

EXAMPLES : >ERA MYFILE.HBA

ERA may also be used with wildcards as follows:

>ERA MY*.*

will erase any files starting with 'MY' and any extension.

REMARKS : Each filename will be displayed and the user has the option to enter 'Y' to delete the file or 'N' to retain the file. If wild cards have been specified the next filename is displayed. Due to the nature of the 'RAM disk', the files can at no stage be recovered.

EXECUTE

3.4.3.8 **COMMAND** : EXECUTE

EXPLANATION : Executes the current program in executable RAM page 0.

EXAMPLES : >EXECUTE

>execute

REMARKS : This command will normally only be used after LOADing a program or after using the INP command. However once a program has been loaded and if it does not alter itself in any way, EXECUTE will execute it again without having to re-load the program.

FORMAT

3.4.3.9 **COMMAND** : FORMAT

EXPLANATION : Re-sets HUNTER's filing system.

EXAMPLES : >FORMAT
FORMAT?: Are you sure (Y/N)

REMARKS : When FORMAT is executed all files are deleted from HUNTER's memory, leaving the user with only the built-in system files. This command should be used with caution. It is generally only required if a machine code program loaded into HUNTER has crashed causing destruction of files and directories.

If 'Y' is typed in response to the above question the File Manager will re-set HUNTER's filing system. Any files in the storage area will be lost. Also any programs loaded into the Basic workspace are deleted.

If any other character is typed HUNTER will return to the File Manager.

INP

3.4.3.10 **COMMAND** : INP [n filename]

EXPLANATION : Inputs a file via the serial port.

EXAMPLE : INP can be used to load files in either Intel Hex format, with checksums etc., or as 8-bit data but without any error checking. The second method of loading is, however, faster than using Intel Hex.

(1) Loading Intel Hex files

Files to be loaded must be in Intel Hex format, see section 9.11, HEX DATA FORMAT. If they are command files, overlay files or data files etc., this is achieved with a standard utility reproduced in section 9.5, DEMONSTRATION PROGRAMS.

Typing INP readies HUNTER to receive data, which is put into RAM page 0. The data is not written directly into a file. When completed the number of 256 byte blocks loaded is displayed. It may be copied to a file using the SAVE command, or if it is a program, executed with the Execute command.

(2) Loading 8-bit format.

Data is loaded directly into a specified file in blocks of 256 bytes. The format of the command is:

INP n filename

For example a 14k overlay file:

>INP 56 OVERLAY1.OVR

The communications must have parity set to 8-bit and SERIG (Serial ignore character) set to off.

The number of blocks must be specified because there is no end of file character. If too many blocks are expected, the screen will display 'waiting', and reception may be terminated with 'ESC'.

REMARKS : When HUNTER is loading data the following messages will appear on the screen:

```
>WAITING
>LOADING
>WAITING
>LOADING
.
.
.
.
```

HUNTER responds to each pause in transmission with a 'WAITING' message to indicate that data is not being received.

On completion of Hex loading the number of 256 byte blocks now in memory is displayed.

The number of blocks, n, can be overspecified. For instance:

INP 100 filename

will load most files. Only the number of blocks actually required will be used following termination by 'ESC'.

The number of blocks is calculated by:

1 block = 256 bytes
1K bytes = 4 blocks

NOTE: Because INP clears the receive comms buffer to remove spurious data when it is invoked, it is important to ensure that the HUNTER is ready to receive before the transmitting device begins to SEND.

KEYS

3.4.3.11 COMMAND : KEYS

EXPLANATION : Operation of the 'KEYS' function displays the alternate function key display line. Further operation of the function key will return HUNTER back to the original display line.

EXAMPLES : SET 1:

DIR STAT BAS COMS CLCK TERM EDIT KEYS

SET 2:

DIR STAT INP SAVE TYPE REN SEND KEYS

REMARKS : DIR and STAT are the most commonly used keys and appear in both sets at easily accessible locations.

DIR - Directory
STAT - File status
BAS - Basic interpreter
COMS - Communications parameters
CLCK - Set clock
TERM - Terminal emulation
EDIT - Text Editor
INP - Load a file
SAVE - Create a file from memory
TYPE - Display an ASCII file
REN - Rename a file
SEND - Transmit files

For further information on soft keys, see section 4.5.4.1, SOFT KEYS.

LOAD

3.4.3.12 **COMMAND** : **LOAD** filename

EXPLANATION : The file is loaded into RAM page 0 ready for execution (EXECUTE) or dumping through the serial port in Hex format (SEND).

EXAMPLES : >LOAD PROG1.COM

The program PROG1.COM is loaded into the execution memory.

REMARKS : Files are loaded into location 100H upwards. Normally, this facility will be used to dump the file in Intel Hex format.

REN

3.4.3.13 **COMMAND** : **REN** filename1=filename2

EXPLANATION : Rename the filename2 to filename1

EXAMPLES : >REN NEW.NAM=OLD.NAM

This will rename the file OLD.NAM to NEW.NAM .

REMARKS : Care should be exercised when renaming files to prevent duplicate file names existing.

SAVE

3.4.3.14 **COMMAND** : **SAVE** n filename

EXPLANATION : Create a file from current memory.

EXAMPLE : >SAVE 128 BIGFILE.COM

This will create a file 32k bytes in length.

REMARKS : A program currently residing from 100H in RAM page 0 is transferred to the specified file. The number of 256 byte blocks also has to be specified.

Calculate the number of blocks 'n' as follows:

1 block = 256 bytes

1 K bytes = 4 blocks

2 K bytes = 8 blocks

8 K oytes = 32 blocks

16 K bytes = 64 blocks etc.,

The maximum program size is 54K.

See 'LOAD' in Part 5, BASIC FUNCTIONS.

SEND

3.4.3.15 **COMMAND** : **SEND** filename [A]

EXPLANATION : Outputs files via the serial port.

EXAMPLES : To output a file in 8-bit format:

>SEND DEMO.COM

To output an ASCII file to, say, a printer:

>SEND LETTER.TXT A

To output in Intel Hex format, first LOAD the file into the execution RAM page 0, then type:

SEND n

n is the number of 256 byte blocks required to be sent.

REMARKS : To transmit in 8-bit format, remember to set the communication parameters correctly i.e. Parity to 8-bit and SERIG to off.

Transmission can be aborted with the 'ESC' key.

The only real difference between 8-bit format and ASCII dumping is that transmissions cease on the first occurence of the end of file character 'control Z' (1A Hex) in the file.

NOTE: If transmitting, using 8-bit data format, a .COM or a HUNTER.HBA file, the LF option in the communications menu **MUST** be turned off to prevent the insertion of LF characters following CR characters, possibly corrupting the object code file being received.

NOTE: SEND cannot transmit a file whose name begins with a numeric character and will return a Syntax error. Rename the file using REN first, or use Hex format.

STAT

3.4.3.16 **COMMAND** : **STAT** [filename]

EXPLANATION : Extended file directory display.

EXAMPLES : >STAT FIRSTFIL.COM

The following will be displayed:

```
 Recs  Bytes  Ext.  ACC
  64    16K    1    R/W   :FIRSTFIL.COM

 24K bytes available
```

Recs = Number of 128 byte blocks.
Bytes = Size of file in 'K' bytes.
Ext = Number of 16K extension blocks.
Acc = Access to file R/W - Read Write R/O - Read Only.

REMARKS : STAT may also be used with wildcards.

>STAT *.HBA

Will display the status of all HUNTER Basic files stored in memory.

This page intentionally left blank

TERM

3.4.3.18 **COMMAND** : TERM

EXPLANATION : HUNTER emulates a CRT terminal.

EXAMPLES : >TERM

REMARKS : In terminal emulation mode, HUNTER can be used as:

1) A remote dialup terminal.

2) A peripheral to other computer systems.

3) A portable 'Telex'.

See section 6.8, TERMINAL EMULATION, for further information.

Two function keys are implemented in TERM:

Function 2 - COMMS set up
Function 8 - Exit to system

The type of terminal to be emulated is determined by the setting of the communication parameters. See section 6.4.1.

NOTE: Automatic power down does not operate in Terminal Emulation. This prevents inadvertant dropping of communications lines.

TERM may be used from user programs (see section 6.8.8).

TYPE

3.4.3.19 **COMMAND** : TYPE filename

EXPLANATION : List ASCII file to HUNTER's screen.

EXAMPLES : >TYPE FILE.TXT

This will output the filename FILE.TXT to the screen.

REMARKS : Scrolling may be halted with 'control S' and restored with any other key. Any character typed while the screen is scrolling, besides 'control S' will abort the display.

Typing non-ASCII files may give confusing results.

CP/M INTERFACE

3.5 CP/M provides a universal interface linking application programs into the operating system. Through a system of calls to one location (0005 Hex) with the CPU register C set to the call number, the ROM-based operating system in HUNTER will implement the function and return to the user program. HUNTER is therefore compatible with many readily available commercial products, conforming to these standard system calls.

3.5.1 SYSTEM CALLS

CP/M system calls fall into two distinct categories. They are console (e.g. keyboard/LCD display) I/O and file I/O. Files are organised as CP/M disk files and may be accessed through a file control block (FCB).

TABLE 3.2 SYSTEM CALLS

0 System Reset	19 Delete File
1 Console Input	20 Read Sequential
2 Console Output	21 Write Sequential
3 Reader Input	22 Make File
4 Punch Output	23 Rename File
5 List Output	24 Return Login Vector
6 Direct Console I/O	25 Return Current Disk
7 Get I/O Byte	26 Set DMA Address
8 Set I/O Byte	27 Get Addr(Alloc)
9 Print String	28 Write Protect Disk
10 Read Console Buffer	29 Get R/O Vector
11 Get Console status	30 Set File Attributes
12 Return Version Number	31 Get Addr(Disk Parms)
13 Reset Disk System	32 Set /Get User Code
14 Select Disk	33 Read Random
15 Open File	34 Write Random
16 Close File	35 Compute File Size
17 Search for First	36 Set Random Record
18 Search for Next	37 Reset Drive

System calls 38-47 refer to HUNTER operating system only.

CP/M is a registered trademark of Digital Research Corp.

3.5.2 FILE CONTROL BLOCK

The FCB contains 36 bytes of information on the current file being accessed. On HUNTER, the default FCB location is at 005CH, containing the following information:-

TABLE 3.3 FCB DESCRIPTION.

Byte	Description
1	Drive code (zero)
2-9	Eight Character ASCII filename
10-12	ASCII file extension
13	Current extent number
14-15	Reserved
16	Record count for extent
17-32	Reserved for system use
33	Current record in sequential file operation
34-36	Random record number

3.5.3 HUNTER SYSTEM CALLS

This manual does not contain an exhaustive description of each system call, since they are defined and explained in great depth in many other publications. However, there does follow a brief description of each call, together with details of extra calls specific to HUNTER.

The C register is loaded with the call number plus any other parameters into other registers, and the A call is made to address 0005. Registers not mentioned are irrelevant on entry, and none are necessarily preserved on exit. For making these calls from Basic, see the CALL and ARG in Part 5, BASIC FUNCTIONS and section 4.8, MACHINE CODE CALLS.

3.5.3.1 **FUNCTION** : System Reset
CALL NO. : 0 {ie Register C = 0 }
ENTRY : None
EXIT : None
REMARKS : A system restart, re-entering HUNTER's operating system.

3.5.3.2 **FUNCTION** : Console Input
CALL NO. : 1
ENTRY : None
EXIT : A=ASCII character.
REMARKS : Read keyboard and echo onto screen. The call will wait until a key is pressed.

3.5.3.3 **FUNCTION** : Console Output
CALL NO. : 2
ENTRY : E=ASCII character.
EXIT : None
REMARKS : The character is displayed on the screen at the location of the cursor.

3.5.3.4 **FUNCTION** : Reader Input
CALL NO. : 3
ENTRY : None
EXIT : A=ASCII character.
REMARKS : The next character is read from the serial I/O buffer.

3.5.3.5 **FUNCTION** : Punch Output
CALL NO. : 4
ENTRY : E=ASCII character.
EXIT : None
REMARKS : The character is sent out through the serial port.

3.5.3.6 **FUNCTION** : List Output
CALL NO. : 5
ENTRY : E=ASCII character.
EXIT : None
REMARKS : As call 4.

3.5.3.7 **FUNCTION** : Direct Console I/O
CALL NO. : 6
ENTRY : E=0FFH for input, E=ASCII character for output.
EXIT : A=ASCII character if input, A=status if output.
REMARKS : Similar to calls 1 and 2, but there is no automatic echo to the screen. If a character is not ready from the keyboard then A=00 on return.

3.5.3.8 **FUNCTION** : Get I/O Byte
CALL NO. : 7
ENTRY : None
EXIT : A=I/O status.
REMARKS : The present I/O status byte is returned.

3.5.3.9 **FUNCTION** : Set I/O Byte
CALL NO. : 8
ENTRY : E=I/O value.
EXIT : None
REMARKS : I/O byte is set.

3.5.3.10 **FUNCTION** : Print String to Console
CALL NO. : 9
ENTRY : DE=string address.
EXIT : None
REMARKS : An ASCII string whose start address is loaded into DE, is displayed on the screen. The string must be terminated with "$".

3.5.3.11 **FUNCTION** : Read Console Buffer
CALL NO. : 10
ENTRY : DE=buffer address.
EXIT : None
REMARKS : The keyboard is read and the characters are placed into a buffer, the start address having been loaded into DE. The first byte contains the maximum number of characters to read into the buffer, the second byte contains the number of characters that have been placed into the buffer when input has been terminated with C/R or L/F. See section 9.15, read console buffer, for further information.

3.5.3.12 **FUNCTION** : Get Console Status
CALL NO. : 11
ENTRY : None
EXIT : A=console status.
REMARKS : If a character has been typed in then 01H is returned, 00H if not.

3.5.3.13 **FUNCTION** : Return Version Number
CALL NO. : 12
ENTRY : None
EXIT : HL=version number.
REMARKS : HUNTER is CP/M version 2.2 compatible, so 0022H is returned.

3.5.3.14 **FUNCTION** : Reset Disk System
CALL NO. : 13
ENTRY : None
EXIT : None
REMARKS : The filing system is reset, and the DMA address is returned to 80H.

3.5.3.15 **FUNCTION** : Select Disk
CALL NO. : 14
ENTRY : None
EXIT : None
REMARKS : HUNTER has "RAM disks" so this call produces no action.

3.5.3.16 **FUNCTION** : Open File
CALL NO. : 15
ENTRY : DE=FCB address.
EXIT : A=directory code
REMARKS : The file specified by the FCB is opened. If successful A=0,1,2 or 3. If not, then A=0FFH.

3.5.3.17 **FUNCTION** : Close File
CALL NO. : 16
ENTRY : DE=FCB address.
EXIT : A=directory code.
REMARKS : If successfully closed, A=0,1,2 or 3, otherwise A=0FFH.

3.5.3.18 **FUNCTION** : Search for First
CALL NO. : 17
ENTRY : DE=FCB address.
EXIT : A=directory code.
REMARKS : The directory is searched for the first match of the FCB. A=0FFH if there is no match, and 0,1,2 or 3 if there is. The filename is returned into the current DMA buffer. The search function returns the complete entry from the buffer.

3.5.3.19 **FUNCTION** : Search for Next
CALL NO. : 18
ENTRY : None
EXIT : A=directory code.
REMARKS : Similar to CALL 17, except that the next entry is looked for. The search function returns the complete entry from the buffer.

3.5.3.20 **FUNCTION** : Delete file
CALL NO. : 19
ENTRY : DE=FCB address.
EXIT : A=directory code.
REMARKS : Files which match the FCB are deleted. If none exist A=0FFH is returned.

3.5.3.21 **FUNCTION** : Read Sequential
CALL NO. : 20
ENTRY : DE=FCB address.
EXIT : A=directory code.
REMARKS : The specified file is read, and the sector copied to the current DMA address. A = 00 for a successful read.

3.5.3.22 **FUNCTION** : Write Sequential
CALL NO. : 21
ENTRY : DE=FCB address.
EXIT : A=directory code.
REMARKS : The record at the current DMA address is written to the disk. A=00 to indicate a successful write.

3.5.3.23 **FUNCTION** : Make File
CALL NO. : 22
ENTRY : DE=FCB address.
EXIT : A=directory code.
REMARKS : The specified file is created. A=0-3 if successful, or A=0FFH if not.

3.5.3.24 **FUNCTION** : Rename a File
CALL NO. : 23
ENTRY : DE=FCB address.
EXIT : A=directory code.
REMARKS : The file specified in the first 16 bytes of the FCB is renamed to that name in the second 16 bytes.

3.5.3.25 **FUNCTION** : Return Login Vector
CALL NO. : 24
ENTRY : None
EXIT : HL=login vector.
REMARKS : HUNTER returns HL=1 and A=1, indicating a single disk system.

3.5.3.26 **FUNCTION** : Return Current Disk
CALL NO. : 25
ENTRY : None
EXIT : A=current disk.
REMARKS : Zero is always returned on HUNTER.

3.5.3.27 **FUNCTION** : Set DMA Address
CALL NO. : 26
ENTRY : DE=DMA address.
EXIT : None
REMARKS : The current disk data buffer area is set to DE.

3.5.3.28 **FUNCTION** : Get Allocation Address
CALL NO. : 27
ENTRY : None
EXIT : HL=allocation address.
REMARKS : Zero only is returned.

3.5.3.29 **FUNCTION** : Write Protect Disk
CALL NO. : 28
ENTRY :
EXIT :
REMARKS : Returns with no effect.

3.5.3.30 **FUNCTION** : Get Read Only Vector
CALL NO. : 29
ENTRY :
EXIT :
REMARKS : Returns with no effect.

3.5.3.31 **FUNCTION** : Set File Attributes
CALL NO. : 30
ENTRY : DE=FCB address.
EXIT : A=directory code.
REMARKS : File attributes for read only etc. are set or reset as specified.

3.5.3.32 **FUNCTION** : Get Disk Parameter Address
CALL NO. : 31
ENTRY : None
EXIT : HL=disk parameter block address
REMARKS : Disk parameters are returned which indicate a double sided double density disk. This is to allow for expansion within HUNTER. It should not be used for space calculation purposes.

3.5.3.33 **FUNCTION** : Get or Set User Code
CALL NO. : 32
ENTRY :
EXIT :
REMARKS : Returns with no effect.

3.5.3.34 **FUNCTION** : Read Random
CALL NO. : 33
ENTRY : DE=FCB address
EXIT : A=directory code.
REMARKS : The file is read at the specified record. Normally non zero directory codes mean an error.

3.5.3.35 **FUNCTION** : Write Random
CALL NO. : 34
ENTRY : DE=FCB address.
EXIT : A=directory code.
REMARKS : The file is written at the specified record. A=0 means a successful write.

3.5.3.36 **FUNCTION** : Compute File Size
CALL NO. : 35
ENTRY : DE=FCB address
EXIT : Random record field set to one plus the last record.
REMARKS :

3.5.3.37 **FUNCTION** : Set Random Record
CALL NO. : 36
ENTRY : DE=FCB address.
EXIT : Random record field set.
REMARKS :

3.5.3.38 **FUNCTION** : Reset Drive
CALL NO. : 37
ENTRY : DE=drive vector.
EXIT : None
REMARKS : HUNTER's filing system is reset.

3.5.3.39 **FUNCTION** : Set up the RS-232 Serial Port
CALL NO. : 38
ENTRY : None
EXIT : None
REMARKS : The software setup routine is entered to set up the serial communications.

3.5.3.40 **FUNCTION** : Initialise the Clock
CALL NO. : 39
ENTRY : None
EXIT : None
REMARKS : The time and date buffer RAM is copied to the clock.

3.5.3.41 **FUNCTION** : Write Random with zero fill
CALL NO. : 40
ENTRY : DE=FCB address
EXIT : A = directory code
REMARKS : The write random with zero fill operations is similar to function 34, with the exception that a previously unallocated block is filled with zeros before data is written.
NOTE: on early HUNTERs, this call was used for a different function.

3.5.3.42 **FUNCTION** : Initialise communications
CALL NO. : 41
ENTRY : None
EXIT : None
REMARKS : This call allows configuration of the communications exactly as the standard HUNTER initialise communications routine.

3.5.3.43 **FUNCTION** : Serial Status
CALL NO. : 42
ENTRY : None
EXIT : A = port status
REMARKS : To test for pending serial input characters A=0 for no characters read else A is non-zero

3.5.3.44 **FUNCTION** : Terminal emulation call
CALL NO. : 43
ENTRY : None
EXIT : None
REMARKS : Enters terminal emulation mode

3.5.3.45 **FUNCTION** : Wand input call
CALL NO. : 44
ENTRY : DE=Buffer Address
EXIT : None
REMARKS : The Wand code is selected previously by setting the bartype location according to wand code being read.

The code is read into the buffer. The user must provide the buffer address and, in the first location of the buffer, the user must provide the maximum number of characters to be read into the buffer. On exit, the second byte of the buffer contains the number of characters that have been placed in the buffer, followed by the wand character's input.

3.5.3.46 **FUNCTION** : Fetch key without echo
CALL NO. : 47
ENTRY : None
EXIT : A = ASC code of character
REMARKS :

3.5.3.47 **FUNCTION** : Read the Clock
CALL NO. : 48
ENTRY : None
EXIT : None
REMARKS : The time and date is copied from the clock into the time and date buffer RAM (see section 9.7), locations TENTHSEC to TENYR.

LOADING FILES

3.6 The most important facility of a portable computer is its ability to communicate with other computers and peripheral devices. HUNTER has an industry standard RS-232 interface, supporting the most common and some of the not so common protocols and baud rates. See Part 6, COMMUNICATIONS.

HUNTER is supported by a wide range of micro computer systems including the IBM-PC, Superbrain, Apple (with Z80 card) and ACT-Sirius. Sections 3.6.4 and 3.6.5, give a detailed description of how to connect the Apple and IBM-PC micro computers to HUNTER and transfer files.

This section describes how to load a program file, an overlay file and an ASCII text file.

Two methods of loading files are described. The first involves loading direct into a file in the 8-bit format. Files are sent out from the host computer as they exist on disk, but, for example, if the PIP utility is used it must have the [O] option to indicate that all 8 bits must be sent. The second method is more time consuming, but has the advantage of built in checksums to guarantee the accuracy of data. All files, including ASCII data files, must be converted into Intel Hex format, see section 9.10, HEX DATA FORMAT. These new files may be PIPed out to HUNTER in the normal manner. Both methods are described in full detail in the following subsections. The examples are particular to a CP/M machine.

NOTE: Some CP/M systems may not be configured for 8 bit communication. Check the serial port hardware manual for details of the UART configuration if in doubt.

Apple is a registered trademark of Apple Computer Inc.
IBM-PC is a registered trademark of IBM Corp.
Superbrain is a registered trademark of Intertec.
ACT-Sirius is a registered trademark of ACT.

3.6.1 FILE DESCRIPTION

File Name	Type	Length(k)	No.blocks (4xlength)	No.records (8xlength)
MAIN.COM	Object	22k	88	176
OVERLAY.OVR	Object	10k	40	80
PRODUCTS.TXT	ASCII	14k	56	112

The program MAIN.COM is a fictional file which organises and sorts a database file PRODUCTS.TXT . The file OVERLAY.OVR is an overlay file which like MAIN.COM is an executable program, but parts are loaded into the execution RAM page as and when required by MAIN.COM. All three files must be present in the filespace for correct operation.

3.6.2 LOADING METHOD 1: 8-BIT DATA

Step 1 Set up the communications, see Part 6, COMMUNICATIONS.

```
Baud rate : As required.
Protocols : None.
Parity    : 8-bit.
RTS       : As required.
SERIG     : Off.
```

Step 2 Determine the size of the files to be copied.

```
A>STAT *.*

Recs  Bytes  Ext  Acc
176    22k    2   R/W  A:MAIN.COM
80     10k    1   R/W  A:OVERLAY.OVR
112    14k    1   R/W  A:PRODUCTS.TXT
```

HUNTER loads files in blocks which are each twice the size of Records. Therefore MAIN.COM is 88 blocks in length.

Step 3 Load MAIN.COM

Connect up HUNTER to the host computer with a suitable 5 way RS-232 cable. Then on HUNTER type:

```
>INP 88 MAIN.COM
```

After typing ENTER, HUNTER will respond with

* Waiting

To abort use the ESC key. If PIP is to be used to transfer the file to HUNTER using the LST: printer port, then type on the host computer

PIP LST:=MAIN.COM[O]

The [O] suffix is very important because this tells PIP to list using the full 8 bits. HUNTER will respond to the beginning of transmission with the message

* Loading

If too many blocks have been specified in the INP command then after transmission has completed the 'waiting' message will again be displayed. In this case use the ESC key to complete the transference of the file.

The file MAIN.COM will now reside in a HUNTER file.

Step 4 Load OVRERLAY.OVR

The sequence of commands for loading this file is as follows:

HUNTER >INP 40 OVERLAY.OVR

Host PIP LST:=OVERLAY.OVR[O]

Step 5 Load PRODUCTS.TXT

This file is loaded with these commands:

HUNTER >INP 56 PRODUCTS.TXT

Host PIP LST:=PRODUCTS.OVR[O]

Step 6 Run the program

The three files MAIN.COM, MAIN.OVR and PRODUCTS.TXT should all now be in HUNTER files. Check this with the DIR command. Execute the CP/M program with

>MAIN

NOTE: Some CP/M machines may not be capable of transmitting 8 bits per byte.

3.6.3 LOADING METHOD 2: INTEL HEX DATA FORMAT

Step 1 All files must be converted into Intel Hex data format. This is achieved with a standard CP/M utility program (usually called UNLOAD) a version of which is reproduced in section 9.5.4, PROGRAM 4 'UNLOAD'.

When converted you will have two copies of each file, one in Intel Hex format:

MAIN.COM	MAIN.HEX
OVERLAY.OVR	OVERLAY.HEX
PRODUCTS.TXT	PRODUCTS.HEX

Step 2 Set up the communications, see Part 6, COMMUNICATIONS.

Baud rate	: As required
Protocol	: NONE
Parity	: As required
RTS	: As required
SERIG	: OFF

Step 3 Load MAIN.COM

Connect up HUNTER to the host computer with a suitable 5 way RS-232 cable. Then on HUNTER type:

>INP

After typing ENTER, HUNTER responds with

* Waiting

To abort use the ESC key. If PIP is to be used to transfer the file to HUNTER using the LST: printer port, then type on the host computer

PIP LST:=MAIN.HEX

As soon as transmission begins, HUNTER responds with

* Loading

The file is loaded not directly into a file, but to RAM page 0, the program execution page. When the file is loaded and HUNTER displays

```
* Loading completed
File length = 88 Blocks
```

the data is written into a file with the SAVE command. In this case.

```
>SAVE 88 MAIN.COM
```

The file MAIN.COM is now created.

Step 4 Load OVERLAY.OVR

The sequence of commands for loading this file is as follows:

```
HUNTER    >INP

Host      PIP LST:=OVERLAY.HEX

HUNTER    >SAVE 40 OVERLAY.OVR
```

Step 5 Load PRODUCTS.TXT

This file is loaded with these commands:

```
HUNTER    >INP

Host      PIP LST:=PRODUCTS.HEX

HUNTER    >SAVE 56 PRODUCTS.TXT
```

Step 6 Run the program

The three files MAIN.COM, MAIN.OVR and PRODUCTS.TXT should all now be in HUNTER files. Check this with the DIR command. Execute the CP/M program with

```
>MAIN
```


NOTE: This method of loading HUNTER is again suitable for all types of files, but compared with 8-bit loading is two and a half times slower. However it does have the advantage that if an error occures during transmission then the hex checksum will detect this and

```
* Loading error
```

will be displayed.

3.6.4 HUNTER TO APPLE INTERFACE

An Apple IIe micro computer fitted with a Z80 card and running the CP/M operating system can communicate with HUNTER via the RS-232 communications port.

When the HUNTER is downloading to the Apple, problems may arise if the file being transferred is larger than 16K. This is because the Apple does not prevent incoming data during a disk access, and therefore may well loose characters. It is advisable to arrange for data within the HUNTER to be stored in 15K files and then transmit each file separately.

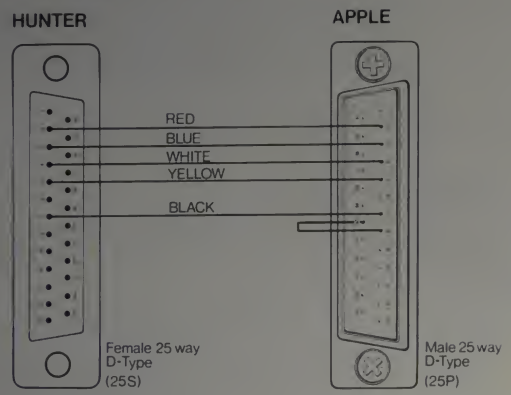
3.6.4.1 Apple Serial Card

A standard Apple compatible serial interface card, preferred type CCS7710A, must be installed in slot 2 of the Apple with its 25 pin 'D' connector mounted on the rear panel in position 7.

3.6.4.2 Interface Cable

The interface cable required to connect HUNTER to an Apple micro computer fitted with the above serial card should be fitted with a female 25 pin 'D' connector at one end, HUNTER, and a male 25 pin 'D' connector at the other, Apple. Pins 2,3,4,5 and 7 should be connected with a jumper connection from pin 8 to pin 20 on the male connector, Apple, see Fig 3.4, HUNTER TO APPLE CABLE.

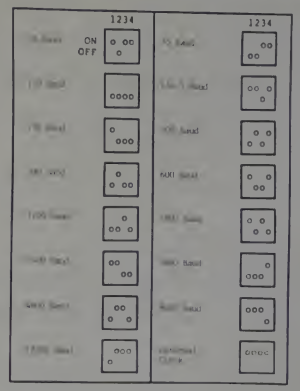
Fig 3.4 HUNTER TO APPLE CABLE



3.6.4.3 **Apple Baud Rates**

The baud rate on the Apple is selected by setting the 4-way DIP switch located at the top right hand corner of the above serial interface card, the switch should be set to 4800 baud, see Fig 3.5, APPLE BAUD RATE SELECTION, for switch setting's.

Fig 3.5 APPLE BAUD RATE SELECTION



3.6.4.4 **HUNTER Baud Rate and Protocols**

HUNTER's communication parameters should be set as follows:

Transmit	Receive
Rate-4800	Rate-4800
Prtcl-N	Prtcl-N
Prty-N	Prty-N
CTS-Y	RTS-HOLD
DTR-N	DSR-N
LF-Y	DCD-N
Echo-Y	Serig-off
Null-0	

See section 6.4, COMMUNICATIONS PORT SOFTWARE, for further details of selecting and setting HUNTER's communication protocols.

3.6.4.5 Basic Program Transfer

When the correct communication protocols have been set and the interface cable installed, program transfer can take place. HUNTER will be treated as RDR: for input and PUN: for output. HUNTER should be in Basic interpreter mode, select 'BAS' from the File Manager function keys, with the program resident in RAM page 0, for Basic text transfers. For file or CP/M program transfers, refer to the examples given earlier.

HUNTER to Apple

To transfer a program from HUNTER to the Apple proceed as follows:

1) Apple >PIP A:filename=RDR:
2) HUNTER LLIST
3) When transmission is complete type LOPCHR26 (control Z) on HUNTER.
4) Apple will now have a copy of the program stored on disk with the specified filename.

NOTE: If the program is greater then 20k the Apple may store part of the file to make more memory available for the incoming program.

Files created on the Apple in this way can be handled with the usual CP/M utilities such as 'ED' or more sophisticated programs such as word processors etc.

Apple to HUNTER

To transfer a program from an Apple disk file to HUNTER proceed as follows:

1) Clear HUNTER's memory with 'NEW'.
2) HUNTER LLOAD
3) Apple >PIP PUN:=A:filename [E]
4) When transmission is complete switch HUNTER off and back on again, the program may now be run.

3.6.5 Hunter to IBM-PC Interface

An IBM PC or XT computer, fitted with the asynchronous communications adaptor can communicate with the HUNTER to either load or unload programs and data.

A package consisting of cable and floppy disk with communications software is available from Husky Computers Limited.

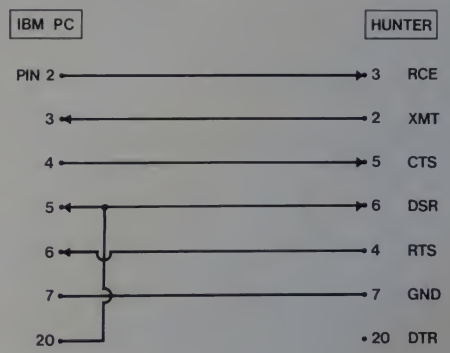
Order as 'IBM-PC Communications Package'.

3.6.5.1 IBM-PC Communications Adaptor

The asynchronous communications adaptor for the IBM machine is the standard part number 1502074. It is available from all IBM-PC dealers and arrives complete with installation instructions for RS-232.

3.6.5.2 Interface Cable

The interface cable required to connect HUNTER to the IBM-PC, fitted with the above serial card, is a special one to ensure correct operation. The two female 25-pin 'D' type plugs are connected as below:



3.6.5.3 Communications Software

a) **The HUNTER**

No special software is required on the HUNTER. However, knowledge of the following commands will be necessary for communications.

From the filemanager:

(i) INP
(ii) SEND

and the following HUNTER BASIC commands/statements:

(i) LLOAD
(ii) LLIST
(iii) LINPUT
(iv) LPRINT
(v) LINCHR
(vi) LOPCHR

See also the section on Communications, Section 6, with particular attention to comms parameter setup.

b) **The IBM-PC**

Special software is required for full communication capabilities on the IBM-PC. This software, together with the interface cable, is available from Husky Computers Limited on an IBM disk.

3.6.5.4 **HUNTER Communications Set-up**

To communicate in both directions at 4800 baud, the communication parameters should be set as follows:

TRANSMIT		RECEIVE	
Rate	- 4800	Rate	- 4800
Prtcl	- None	Prtcl	- None
Prty	- 8-bit	Prty	- 8-bit
CTS	- y	RTS	- hold
DTR	- n	DSR	- n
ECHO	- n	DCD	- n
LF	- n	Serig	- Off
Null	- 0	T/O	- no
T/O	- no		

The above setup allows full bi-directional communications with hardware handshaking. Both ASCII text files and 8-bit object code files can be up and down loaded.

3.6.5.5 **IBM-PC Communications Program**

The files supplied on disk are:

COMMS1.BAS
BASIC.COM
HUNTCOM.BAT

To run the program, follow these steps:

a) Turn on the IBM-PC
b) Insert a 'DOS' disk into the A drive
c) Insert the supplied disk into the B drive
d) Copy the files onto the A disk:

```
copy B:*.* A:
```

e) Remove the B disk and store in a safe place
f) Type:

```
huntcom
```

g) The IBM comms program is now running and asking for your selection of baud rate.

It displays:

```
Asynchronous Communications Program
select baud rate ('s' for system):

    (1) 300
    (2) 1200
    (3) 4800
```

3.6.5.6 **Loading files from the IBM-PC**

Make sure the file you want to send is on one of the disks in the IBM-PC and also that the three communications files are on the disk in the A slot.

a) Plug the cable into the HUNTER and into the asynchronous port.
b) Switch on the HUNTER and enter the file manager.
c) Set up the HUNTER comms parameters.
d) On the HUNTER, type:

```
INP no.blocks filename
```

The number of blocks (256 bytes) may be greater than the actual number, but never less.

e) On the IBM-PC, select the baud rate:

i.e. type '3' for 4800 baud

f) You are now prompted with:

(R)receive or (T)ransmit _

Type 't' for transmit.

g) Enter the filename of the file you wish to transfer. It may be preceded with a drive name if required:

e.g. b:filename

h) The bottom line of the display should show:

transmitting

to indicate that transmission is taking place.

When the file has been sent, the comms program will return to the baud rate set-up. If no more communications are required, type 's' for system.

NOTE: If the message 'waiting to transmit' is displayed, then the HUNTER is not ready to receive the file.

3.6.5.7 Sending files to the IBM-PC

Ensure that the 3 communications files are on the A disk in the IBM.

a) Plug the cable into the IBM and the HUNTER and switch on the HUNTER.
b) It is important to ensure that the RS-232 interface is powered up on the HUNTER before transmission occurs - to avoid an extraneous character on power up wich would otherwise be added to your file. Power up the interface by selecting TERMINAL mode then return to the filemanager.

This procedure only has to be followed when the HUNTER is first powered up.
c) Set up the HUNTER comms parameters as before.

d) On the IBM, select baud rate.

i.e. type '3' for 4800 baud.

e) When prompted with:

(R)eceive or (T)ransmit _

type 'r' for receive. The following message will then appear on the bottom line.

'any key to abort'

f) On the HUNTER, type:

SEND filename

where filename is the name of the file you wish to transmit.

g) When transmission is over, the HUNTER displays the message:

'COMPLETED'

h) Hit any key on the IBM to close the received file and return to the beginning of the program.

3.6.5.8 Communicating HUNTER Basic programs

A HUNTER Basic program (.HBA suffix) is stored in a tokenised form. These may be transferred to and from the IBM-PC, as described above.

A HUNTER Basic program, when stored on the IBM, is not in ASCII text form and cannot, therefore, be listed or displayed.

If it is desired to transfer the ASCII text of a program, it is necessary to transmit the program from the HUNTER Basic Interpreter as follows:

a) Plug the cable into the IBM and the HUNTER and switch on the HUNTER.
b) It is important to ensure that the RS-232 interface is powered up on the HUNTER before transmission occurs - to avoid an extraneous character on power up wich would otherwise be added to your file. Power up the interface by selecting TERMINAL mode then return to the filemanager. This procedure only has to be followed when the HUNTER is first powered up.

c) Set up the HUNTER comms parameters as before.
d) On the IBM, select baud rate.

i.e. type '3' for 4800 baud.

e) When prompted with:

(R)eceive or (T)ransmit _

type 'r' for receive. The following message will then appear on the bottom line.

'any key to abort'

f) Enter Basic on the HUNTER with the Basic file loaded into the workspace. On the HUNTER type:

LLIST

This will list the Basic program to the IBM for storage in the selected file.

g) Transmission is over when the HUNTER displays:

READY

h) Hit any key on the IBM to close its file.

An ASCII file with Basic Text may be transferred directly into the Basic workspace using the LLOAD command in Basic. This is done as follows:

a) Plug the cable into the HUNTER and into the asynchronous port.
b) Switch on the HUNTER and enter the file manager.
c) Set up the HUNTER comms parameters.
d) On the HUNTER enter Basic and type:

LLOAD

e) On the IBM-PC, select the baud rate:

i.e. type '3' for 4800 baud

f) You are now prompted with:

(R)eceive or (T)ransmit _

Type 't' for transmit.

g) Enter the filename of the file you wish to transfer. It may be preceded with a drive name if required:

e.g. b:filename

h) The bottom line of the display should show:

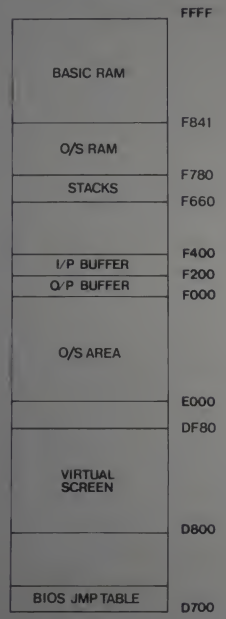
transmitting

to indicate that transmission is taking place.

MEMORY MAPS

3.7 This section describes the operating system RAM at the top of page R0. Important locations are detailed, many of which the user may wish to use for advanced programming. Fig 3.6, RAM PAGE 0, is an overall map of the system. A listing of memory locations may be found in section 9.7, MEMORY LOCATIONS.

Fig 3.6 RAM PAGE 0



AUTO RUN

3.8.1 **INTRODUCTION**

The automatic program continuation command 'CONT' provides the user with the facility to continue execution of a program after HUNTER has been switched off. Power off may have been caused by execution of the POWER OFF function in Basic, Auto power off or operation of the power off key.

3.8.2 **COMMAND SYNTAX**

The syntax of the 'CONT' command is as follows:

>CONT filename

where 'filename' is the name of a file containing a Basic source program or a CP/M program. The three character file type extension does not have to be entered if it is a .COM file, but must be included for HUNTER Basic files. For example, if a Basic program file is called PROG1.HBA, the following command would load and run the program via the Basic interpreter:

>CONT PROG1.HBA

and the CP/M program PROGA.COM would be run if the following was entered:

>CONT PROGA

If the filename does not have a .COM or .HBA extension, 'CONT' will default the file type extension to .COM, so if the filename PROG2.TXT is entered 'CONT' will default the filename to PROG2.COM. If it does not exist a 'File not found' error will occur.

3.8.3 **PROGRAM CONTINUATION**

At any time during program execution HUNTER may be switched off by operation of the power key, automatic power off or any other means *. The next time HUNTER is switched on the program will continue execution from the point at which HUNTER was switched off.

* See 3.8.5.

3.8.4 EXITING 'CONT'

To escape from 'CONT' press (control C) and hold down, then power up. HUNTER will bleep twice and then wait for the escape code to be entered (default code 56580). Failure to enter the correct code will result in a bleep from HUNTER and continuation of the program. If the code is entered correctly then HUNTER will power down immediately. The next power up will revert to the operating system in the normal way. When continuing a .HBA file it is possible to escape from the program by pressing 'ESC' and entering the escape code as above.

3.8.5 POWER LOSS

If power loss occurs by removing the batteries, the next power up will cause HUNTER to reload the file that was being run and execute the program from the start.

Power loss can also be caused by excessive shock, battery failure, nuclear effects or other means.

3.9 DEMOS ERROR MESSAGES

The following are operating system error messages which can occur in DEMOS:

3.9.1 Syntax

Due to:-

a) Missing '=' between 2 filenames in REN
b) Missing number of 128 byte blocks in SAVE
c) Missing space between byte block number and name in SAVE
d) Missing number of 128 byte blocks in INP
e) Missing space between byte block number and name in INP
f) Missing number of 256 byte blocks in SEND
g) Missing CR immediately after the filename in SEND

3.9.2 No File

Due to:-

a) Missing file name in EDIT
b) The file does not exist in ERA
c) The old file name does not exist in REN
d) The Basic file does not exist when trying to load and run the file from DEMOS
e) No files in the HUNTER when enter STAT *.*
f) .COM file does not exist when name is entered

3.9.3 Bad Name

Whenever an invalid file name format is specified.

3.9.4 Memory Overflow

Due to:-

a) The text equivalent of a basic tokenised file exceeding the available space in Ram page 0.
b) Excess number of 128 byte blocks in SAVE
c) Excess number of 256 byte blocks in SEND

3.9.5 Aborted

Due to:-

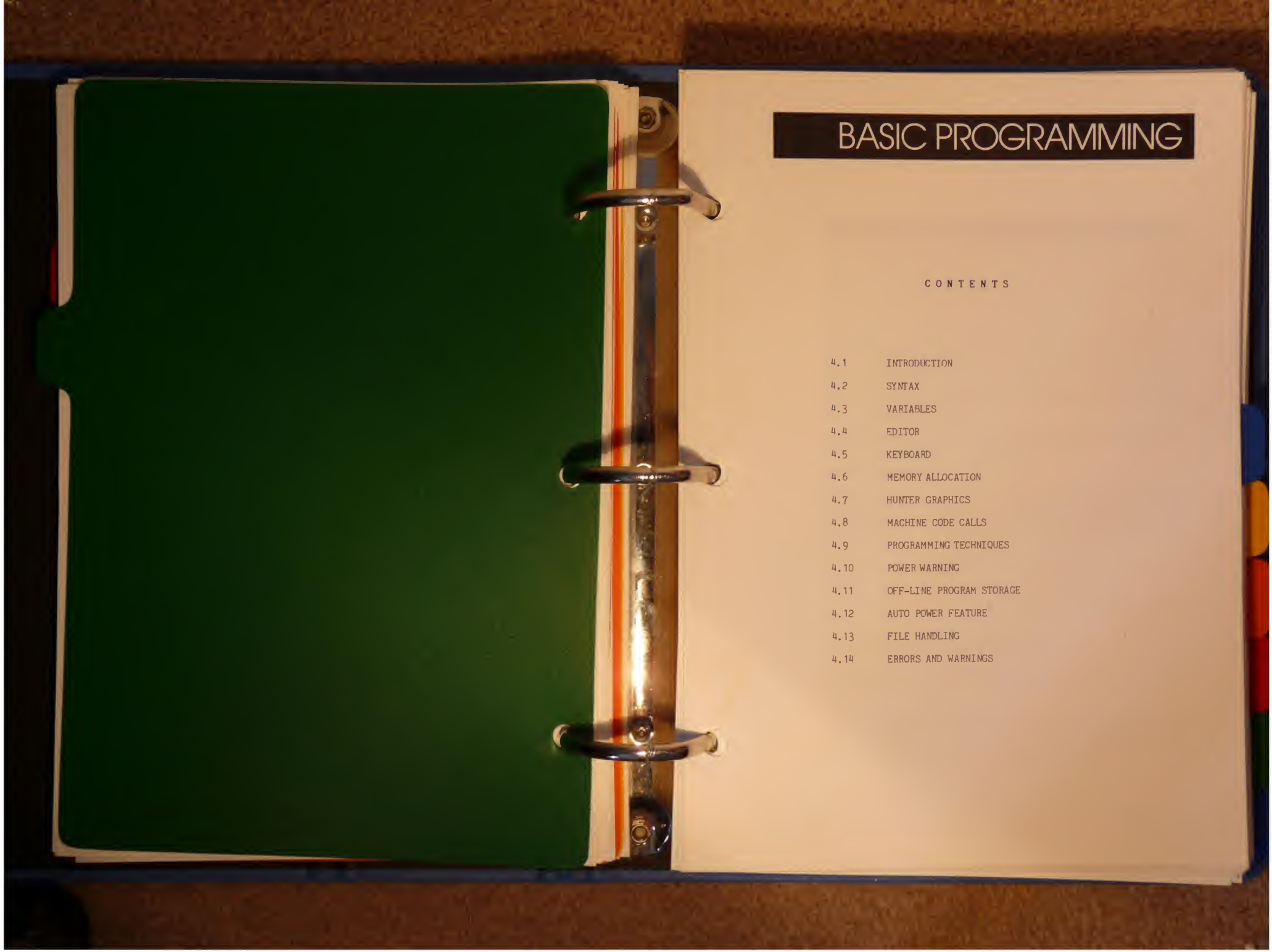
Pressing the ESC key when the HUNTER is executing TYPE or INP

3.9.6 Disk Full

At any time when trying to write to a file when there is no space remaining in the file space.

3.9.7 Disk Error

Due to a file having been corrupted, e.g. an incorrect checksum for the file.



BASIC PROGRAMMING

CONTENTS

INTRODUCTION

4.1 HUNTER Basic is a powerful and flexible Basic interpreter installed within HUNTER's firmware, allowing user programs to be easily written and used without the need for any additional equipment. In addition to normal programming features, HUNTER Basic can also handle communications with other devices.

Because it's designed for use in the portable environment, HUNTER Basic has some unique features not found in other Basics. Principal among these is the ability of HUNTER Basic to keep both programs and data after its power has been switched off. Data can be used later by the same or another program following an intervening period when HUNTER is powered down and not in use at all. Similarly, user programs written in HUNTER Basic are retained and are always available for further use until deliberately cleared.

Different programs can be stored simultaneously in program space or in files and executed independently.

4.1.1 INITIATING HUNTER BASIC

HUNTER Basic is accessed from the File Manager by typing BAS followed by the 'Enter' key or by pressing the appropriate function key if the HUNTER System Functions are displayed. When HUNTER's Basic Interpreter has been entered the following screen is displayed:

```
+-----------------------------------------+
|Hunter Basic Interpreter                 |
|READY                                    |
|                                         |
|                                         |
|                                         |
|File Run Edit List Save Load Kill Syst   |
+-----------------------------------------+
```

To return to File Manager from Basic press 'SYST' function key or type 'SYSTEM' and press the 'Enter' key.

Alternatively, a HUNTER Basic program (.HBA extension) may be executed directly from the File Manager by typing the file name followed by 'Enter'. In this case, the Basic Interpreter screen does not appear and the program executes immediately. For further details see section 3.2, PROGRAM EXECUTION.

4.1.2 **BASIC PROGRAMS**

Program source code for execution by the resident Basic Interpreter can be provided by a variety of sources, including the following:

1) HUNTER Basic (.HBA extension) files.
2) Text (.TXT extension) files.
3) Direct keyboard entry using Edit.
4) Down line loading using LLOAD.

Programs are entered into HUNTER's execution memory by a sequence of line numbers and program statements. Unnumbered lines will execute immediately. Variables can always be inspected using "Print" statements. Effects of functions and other operators can be determined empirically by simply typing unnumbered lines and observing the result.

Certain functions cannot be executed directly
e.g. GOSUB

NOTE: It is recommended that NEW is typed before entering a program in order to remove any previous programs.

4.1.3 **AUTO START BASIC**

Provision has been made for Basic programs to execute immediately on power up. See section 4.12, AUTO POWER FEATURE.

NOTE: A number of programs can be found in section 9.5, DEMONSTRATION PROGRAMS.

SYNTAX

4.2 The functions described in this manual follow the conventions set out below. For all Basic statements and commands the syntax must be correct for the program to run successfully.

1) Basic reserved words are printed in uppercase, but they may be entered into HUNTER as either uppercase or lowercase or a combination of both. Either way, the interpreter will convert them to uppercase.

2) Items not in square brackets are essential arguments to the Basic function and must be included.

3) Square brackets indicate that the item is optional.

4) Punctuation must be included where shown.

4.2.1 **DESCRIPTION OF FUNCTIONS**

HUNTER Basic understands four kinds of function:

Commands

Statements

Operators

Functions

Commands generally initiate an action and can also control execution of application programs.

Statements form the structure of application programs and direct program flow.

Operators are used to relate data and to test for specified conditions.

Functions perform arithmetic or string computations.

Some commands and statements will function without further information, others require an argument. An argument is either a value, an expression or another function.

A complete list of functions is given in section 5.1, INDEX TO BASIC FUNCTIONS.

4.2.2 EXPRESSIONS AND OPERATORS

TABLE 4.1 SYMBOLIC OPERATORS FOR USE WITH NUMERICAL VARIABLES.

=	Equality or assignment of values
+	Addition
-	Subtraction or negation of value
*	Multiplication
^ or **	Powers
/	Division
(	Open parenthesis
)	Closed parenthesis
=>	Equal to or greater than
=<	Equal to or less than
<	Less than
>	Greater than
<>	Not equal to

TABLE 4.2 OPERATORS FOR USE WITH STRING VARIABLES

=	Equality or assignment of values
<>	Not equal to
+	String addition or concatenation

TABLE 4.3 LOGICAL OPERATORS

Logical operators perform logical or Boolean operations on numeric values:

NOT	logical complement
AND	conjunction
OR	disjunction
XOR	exclusive OR
EQV	equivalence
IMP	implication

TABLE 4.4 TRUTH TABLES

NOT

X	NOT X
T	F
F	T

AND

X	Y	X AND Y
T	T	T
T	F	F
F	T	F
F	F	F

OR

X	Y	X OR Y
T	T	T
T	F	T
F	T	T
F	F	F

TABLE 4.4 CONT.

XOR	X	Y	X XOR Y
	T	T	F
	T	F	T
	F	T	T
	F	F	F

EQV	X	Y	X EQV Y
	T	T	T
	T	F	F
	F	T	F
	F	F	T

IMP	X	Y	X IMP Y
	T	T	T
	T	F	F
	F	T	T
	F	F	T

4.2.3 LOGICAL OPERATIONS

Logical operators act upon the binary equivalents of decimal arguments, bit by bit, where the argument is a 16 bit signed 2's complement number i.e. in the range -32768 to +32767. For example:

```
PRINT 10 AND 6
```

will print 2, since:

```
10 = 0000000000001010
 6 = 0000000000000110
```

Performing the AND operation bit by bit, only one bit remains non zero. See the AND truth table in TABLE 4.4 TRUTH TABLES, for further details.

Logical operations may also be used by conditional statements, for example:

```
IF A=B AND B=C THEN PRINT"ALL EQUAL"
```

In this example the following occurs:

1) The variables A,B are tested for equality. If they are equal then a true (-1) condition is returned, if not then a false (0) is returned.

2) The variables B,C are tested for equality. The results are as in (1) above.

3) The logical operation AND is then performed on the results of (1) and (2) above, which returns either a true (-1) or false (0) result. It is this final result upon which the IF statement performs its conditional test (true or false). Since conditional statements always produce a logical result it is possible in some circumstances to omit a relational operator, for example:

```
IF A<>0 THEN PRINT"A IS NON ZERO"
```

may be replaced with:

```
IF A THEN PRINT"A IS NON ZERO"
```

or

```
IF A=5 THEN PRINT -1 ELSE PRINT 0
```

may be replaced with:

```
PRINT A=5
```

since A and 5 are tested for equality and if they are equal the -1 (true) is printed otherwise 0 (false) is printed.

VARIABLES

4.3 The Basic workspace execution memory is automatically partitioned by the interpreter according to the number of program lines entered by the user and the variable storage space required. If more than 54K RAM is required then data must be stored as files which use filespace in other pages of RAM. For further details see section 3.3.5, HUNTER MEMORY ORGANISATION. However arrays of numerals or strings may be dimensioned as normal, the filing operation being totally transparent to the user.

4.3.1 **LINE NUMBERS**

Line Numbers are in the range 0 to 65535. Each line can be up to 254 characters long, including the line number. Extra lines can be inserted in the text by simply typing in a new line number between the two lines of interest. In view of this, it is recommended that line numbers are separated by an increment of 10 for each new line. Line numbers are stored in binary form and occupy a constant space in memory regardless of the number chosen. Each line number requires 2 bytes of memory.

4.3.2 **VARIABLE STORAGE**

HUNTER variables can be stored as either single or double precision floating point numbers. All simple variables are single precision giving 6 digit accuracy. Variables are designated by one alphabetic character (A-Z) followed by an optional alphanumeric character (A-Z) (0-9), giving up to 962 separate variables.

NOTE: Simple variable names ON, TO, IF, LN, AS, OR and PI should be avoided as these are also reserved words. Use as variables can lead to execution errors.

4.3.2.1 **Numerical Arrays**

Additionally, variables may form a one-dimension array using "DIM". Variable arrays may optionally be designated double precision giving 14 digit accuracy but with an additional memory overhead. Single precision arrays are designated by a number in parentheses following the variable name.

A1(19),Z(2),C(X),A(20*T),AD(10),AZ(62)

There can be 962 separately indentified arrays of single precision.
NOTE: The number in parentheses can be an expression. Each variable occupies 5 bytes of HUNTER's user memory in arrays, 7 bytes otherwise.

4.3.2.2 **Double Precision**

Double Precision arrays are indentified by the symbol '!' followed by a number or expression in parentheses. Double precision variables have 14 digit accuracy.

e.g. A1!(19),Z!(Z),C!(X),AA!(23)

These are a further 962 arrays.

Each double precision element occupies 9 bytes of HUNTER memory.

4.3.2.3 **Default Values**

All numeric arrays default to 11 elements unless dimensioned with DIM statement.

4.3.3 **STRING STORAGE**

Single element string arrays are automatically assigned without a DIM statement. The default length is 20 characters.

String names are defined exactly as variable names, with one alphabetic character (A-Z) and one optional alphanumeric character (A-Z,0-9) followed by the '$' symbol giving a total of 962 separately identified string variables.

HUNTER supports strings of 8 bit characters including the full ASCII character set.

The maximum length of any string is 254 characters, with a default size of 20 characters.

```
DIM A1$(10,15)
```

defines a string array of 11 elements, each of which can be up to 15 characters long.

The maximum size of an array depends on the memory available. Examples of String variables are:

```
A1$,Z$,C5$(22),H$(J),MM$(32)
```

String variables can be initialised using LET, just assigned using =(equate), or READing from DATA statements.

The first element of a string array may be referenced with just the variable name, without the need for any number in parentheses.

Example:

A1$,Z$ Actually refer to the variables A1$(0),Z$(0)

A single variable may therefore be defined as:

ST$(0,10) and used as:
ST$

4.3.4 MULTIPLE STATEMENTS

HUNTER Basic supports multiple statements in a single program line. Statements are separated by a colon(:).

Example:

```
100 A=0:B=10:PRINT A,B:B=11
```

The primary use of this feature is to reduce the program size. It may also make a program more legible.

If a colon is used at the end of a line, END is assumed and program execution stops.

The following statements cannot form part of a multiple statement:

DATA
WHILE
WEND

If REM is used in a multiple statement, it must be the last statement in the line.

4.4 EDITOR

An advanced Editor is available in Hunter for creating and modifying text/Basic files. It may also be used for modifying a "Loaded" Basic program.

This may be accessed by typing EDIT as a direct Basic command, or using function key 3.

Basic text may then be edited. To return to Basic, the editor is "Exited". This will re-process the text into the compressed form used by Basic.

A file may also be saved from within the editor.

For a full description of the facilities available within the Editor, see section 7.

KEYBOARD

4.5.1 HUNTER computers are supplied with a standard QWERTY keyboard, see Fig 2.3, KEYBOARD LAYOUT.

4.5.2 **ELECTRICAL RE-DEFINITION**

Keyboard keys are each equated to the characters they represent by unique numerical codes, as listed in section 9.2, ASCII CHARACTER SET. These values are defined according to the American Standard Code for Information Interchange (ASCII).

The value returned to Basic in, for instance, an INCHR statement is the numerical equivalent of the key pressed.

The ASCII definitions of each key for the keyboard are contained in 116 consecutive RAM locations named KEYBUF. The upper shift keys are defined in the first 56 locations, followed by the 56 lower shift keys. These memory locations form a 'map' of the physical keyboard. On power up, the HUNTER automatically defines these locations to a standard configuration. However, the programmer has the option to be able to define special keyboard arrangements.

The schematic of the keyboard, see section 9.6 KEYBOARD MEMORY MAP, shows the value in KEYBUF for each key.

As an example, consider the key fourth from the left on the top row. If it was required to program this key to be $ in upper case and S in lower case, two Poke operations would be required.

```
POKE (KEYBUF),36
POKE (KEYBUF + 56),83
```

It is important to note that on powering up HUNTER, the keyboard will revert to the standard version. The programmer should define the special keyboard requirements at an early stage in the program and ensure that the re-definition occurs each time the program is run.

4.5.3 The codes used by the special control keys are listed in section 2.4.

4.5.4 **FUNCTION KEYS**

There are eight programmable function keys. They may be used as a quick method of entering frequently used commands or functions, or as eight separate interrupt keys.

4.5.4.1 **Soft Keys**

On entering the Basic interpreter, the bottom line of the LCD displays the default values. These may be changed at any time with the KEY function as described in Part 5, BASIC FUNCTIONS. The screen only displays the first four characters, although the string may be up to fifteen characters long.

The display of the soft keys is entirely optional, but they are always active no matter whether they are displayed or not. This feature is controlled with the KEY ON and KEY OFF commands.

4.5.4.2 **Interrupt Keys**

The soft keys may be used to provide interrupts into Basic programs. The relevant Basic statements are:

```
ON KEY (n) GOSUB line No.
KEY (n) ON
KEY (n) OFF
KEY (n) STOP
```

The interrupts are tested at the beginning of every Basic statement. Once a particular key has been pressed and has interrupted, then it cannot interrupt again, although other keys can still cause an immediate interrupt. However, one interrupt on a particular key can be held pending while its interrupt routine is executing.

MEMORY ALLOCATION

4.6.1 PROGRAM LIMITS AND MEMORY USAGE

1. **Ranges**

Variables:	±9.99999E±126
String arrays:	Up to 254 characters per string
Line numbers:	0-65535 inclusive
Program line length:	Up to 254 characters including line number

2. **Precision**
Single precision variables have 6 digit resolution.
Double precision variables have 14 digit resolution.

3. **Memory overhead**
Program lines require 4 bytes minimum, as follows:

Line number:	2 bytes
Line length:	1 byte
Carriage return:	1 byte

Also each reserved word, operator, variable name character, special character and constant character requires 1 byte.
Maximum program size in present versions: 54K Bytes (Page 0)

4.6.2 DYNAMIC (RUN-TIME) MEMORY ALLOCATION

1. **Symbol Table**

Entries occupy: 7 Bytes each
Maximum symbol table size: 16K Bytes

2. **Single Precision Array**

DIM A(X) occupies (X+1)*5+7 Bytes

3. **Double Precision Array**

DIM A1(X) occupies (X+1)*9+7 Bytes

4. **String Array Elements**

DIM A$(X,N) occupies (X+1)*N+7 Bytes

5. **Array Size**

Maximum number of elements in an array is 16,383.

HUNTER'S GRAPHICS

4.7.1

HUNTER supports a graphics display of 240 x 64 pixels. Each point can be set to either white or black with the appropriate statement. Entering graphics mode is via the SCREEN statement, but this may be unnecessary since most of the functions which operate in graphics mode will automatically invoke the graphics screen.

The graphics statements in Basic are:-

CHAR	LOCATE
CIRCLE	POINT
CLS	PRESET
LINE	PSET

SCREEN 1 enters graphics mode but also defaults the current character set to CHAR 0 and clears the display.

The origin of the graphics screen is the top left hand corner, with the co-ordinates (0,0). So PSET(0,0) will set the corner pixel black. The origin for the LOCATE function, to determine the starting point for displaying ASCII characters, is (1,1) in the top left hand corner.

Co-ordinates when specified are always absolute, ranging 0-239 for x and 0-63 for y. If two pairs of co-ordinates are required, then the first pair may be defaulted to the last plotted position.

The above functions can be made to draw characters or lines in either white on black or black on white. This is achieved by appending another parameter after the command. For example, to print inverse characters of medium size use:-

```
CHAR 2,1
```

The '1' means inverse.

A detailed description of the above graphics commands can be found in Part 5, BASIC FUNCTIONS.

NOTE: Graphics mode on the HUNTER does not directly support the simple cursor manipulation codes, e.g: backspace, line feed, etc. The operating system generates the required cursor movements.

4.7.2 GRAPHICS DEMONSTRATIONS

The following program will draw a number of circles across the screen and back again:

```
10  SCREEN 1
20  FOR K=1 TO 167 STEP 4
30  CIRCLE (K+30,32),30
40  NEXT K
50  CLS
60  FOR K=167 TO 1 STEP -4
70  CIRCLE (K+30,32),15
80  NEXT K
90  CLS
100 GOTO20
```

The 'LINE' command may also be used for drawing boxes as well as drawing lines. The command:

```
10 LINE (100,40)-(200,20)
```

will draw a line from the co-ordinates 100,40 to 200,20, the top left hand corner of the screen being 0,0. The following will draw a box with the bottom left hand corner at point 100,20 and the top right hand corner at 200,20:

```
10 LINE (100,40)-(200,20),1,B
```

if the second parameter is used then the first must also be specified. The box may be filled in by appending 'F' to the second parameter.

The Graphic's cursor can be positioned to any point on the screen by using the 'LOCATE' command. Pixels are set by using 'PSET' or cleared by 'PRESET'. The following example will produce a dotted line across the screen:

```
10 SCREEN 1
20 FOR X=0 TO 239 STEP 4
30 PSET (X,30)
40 NEXT X
50 END
```


MACHINE CODE CALLS

4.8 HUNTER is designed to be extremely versatile and flexible in its use. For some applications it is not possible to perform every function using pure Basic routines, usually for reasons of speed. It is possible to write machine code routines in these situations.

Also covered in this section is some general guidelines in the control of HUNTER by means of PEEK and POKE to specified memory addresses.

It should be emphasised that the use of features described here is not recommended for users unfamiliar with computing at machine code level.

4.8.1 THE MACHINE CODE

HUNTER uses the CMOS NSC-800 microprocessor, which provides an instruction set entirely compatible with the popular Z80 microprocessor. For that reason it is not proposed to go into detail about the facilities offered by machine code and users are referred to any of the standard books which cover the Z80.

It is not recommended, however, that the stack pointer register (SP) is used in a fashion incompatible with supporting interrupts. Standard use in the form of subroutine CALLs, PUSHs and POPs, however, is supported.

4.8.2 BASIC CALL

To execute a machine code program the function:-

```
CALL(addr)
```

is used. This will pass control from Basic to the address. A machine code RET will return the user to Basic.

4.8.3 PASSING SIMPLE PARAMETERS

It is possible to pass information both from Basic to the 'CALLed' program and return the information back to Basic.

NSC-800 is a trademark of National Semiconductor Inc.
Z-80 is a trademark of Zilog Corporation Inc.

To set up outgoing data the ARG reserved word is used, format as shown. The variable will be equated to the value in brackets.

```
A = ARG(5)
```

will cause the NSC800 internal registers C and E to be set up as follows:-

```
C = 5
E = 0
```

The pair are set up as a 16 bit integer, hence:-

```
ARG (10*256+30)
```

causes:-

```
C = 30 and:-
E = 10 (decimal)
```

when CALL is executed.

These registers are used for compatibility with the CP/M style system call structure.

To return data to Basic the program:-

```
1000 X = CALL (addr)
```

causes the value of variable X to take on the value contained in register A.

4.8.4 PASSING MULTIPLE VARIABLES

To pass more than one integer between a 'CALLed' program and Basic the machine code linkage stack is used.

This is controlled in Basic by PUSH and POP. In the routine:-

```
1000 PUSH X,Y
1010 A = CALL (addr)
1020 Y = POP(1)
1030 X = POP(1)
```

The integer values of X and Y are first placed on the stack and made available to the called routine, and then removed by the POP instruction back into their own variables.

When CALL is executed the register pair HL is used as an address pointer to the bottom of the stack on which the variables are placed:-

```
X HIGH
X LOW
Y HIGH
HL->Y LOW
```

In other words HL is addressing the low byte of the most recently PUSHed variable. To access others then HL is incremented. These values may either be used by the called routine or modified and used by Basic. It is not necessary to preserve the contents of HL.

4.8.5 CONTROLLING HUNTER

Section 9.7, MEMORY LOCATIONS and 9.8, PORT ALLOCATIONS, contain list's of available addresses of HUNTER parameters and system call addresses. The parameters may be written or read by means of PEEK and POKE operations. System calls may be used for functions such as the CLOCK or communications parameters.

4.8.6 AVAILABLE MEMORY

By subtle use of the DEFSEG statement, to address the required page, many places in RAM can be used for machine code implants. However, the only place in common RAM available (whatever DEF SEG is set to) is the serial communications buffer. Since the communications are active continuously on receive, take care not to let programs be corrupted. 50 hex bytes are available for small user programs at decimal address 62981 (see section 9.7).

4.8.6.1 MACHINE CODE IN BASIC

The recommended method of storing machine code from BASIC is to store it in a dummy string array, a method for which is outlined below.

a) First calculate the length of the machine code and dimension a **string** array with the required number of single byte elements using the formula:

```
D=(N/2)-1
```

where

N=number of bytes needed for machine code.

and

D=number of single byte elements required in dimension.

To reserve 10 bytes of memory, the required dimension is DIM MC$(4,1).

NOTE: After dimensioning, all the bytes will be set to zero.

b) The start address of the reserved memory must be found. This is done as follows:

Assuming the array name is MC$, the statement AD=VARPTR(MC$)-1 will return the address of the first reserved byte in AD.

c) To actually put the machine code into the array, a series of POKE statements are used, i.e:

```
POKE AD,30,7,14,1,205,5,0,201
```

NOTE: the machine code must be given as decimal numbers.

d) Finally, to execute the machine code, all that is required is

```
X=CALL(AD)
```

where X is a dummy variable and AD is the start address of the machine code as obtained in (b) above.

NOTE: It is vitally important that DEFSEG is set to 0 when the CALL is executed. If DEFSEG is altered in the program, DEFSEG=0 must be executed before the CALL is executed. It is also important that no attempt be made to use the array which is holding the machine code in the Basic program.

4.8.6.2 **PAGING IN MACHINE CODE**

In some applications the user may wish to access other pages from a machine code program. However, this is not a trivial matter and extreme care must be taken when using this facility. The user is recommended to read section 3.3.1 of this manual for full details on the HUNTER memory organisation before proceeding.

In order to use the HUNTER paging, the machine code to switch the page must be located in the common block of ram starting at 49152 (C000H). If the machine code is not in this block of ram, the HUNTER will not be able to access the machine code program once the page has been switched. In order to ensure the machine code is located above 49152 (C000H) the array used to hold the machine code is the last array dimensioned. Check the start

address of the array using VARPTR. If the address returned is still below 49152, use a dummy array to push the start address of the array to above 49152.

The paging control is accessed via an 'OUT (E0H),A' command, with the accumulator containing the required page number with bit 7 set to 1

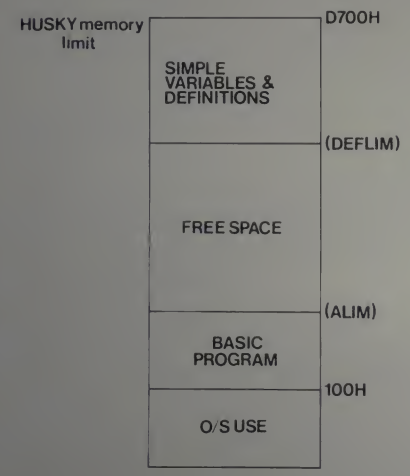
i.e: to switch to page R0, the accumulator should contain 80H (128 decimal).

4.8.7 **AVAILABLE RAM IN R0**

Basic source in HUNTER can extend for 54k, though there must be space for the symbol table.

Fig 4.1, STORAGE USED BY BASIC, shows how program and arrays are stored in lower memory, simple variables and array definitions start at D700 and work down. The gap in the middle is available and unused, the amount being displayed using FRE. The limits are contained in memory locations ALIM and DEFLIM. The address contained at these locations indicates the position of the free space.

Fig 4.1 STORAGE USED BY BASIC



It should be remembered that these locations are allocated dynamically and are entirely dependent upon the program and store. Also, if CLEAR or a new program line is inserted then the data storage is removed, so DEFLIM will point to the top of memory and ALIM to the top of the program. Hence, the program should be RUN before using these values. Clearly, if Basic is only used to start a large machine code program then there will be no variables and only a small amount of Basic source.

4.8.8 **SYSTEM CALLS**

The system calls are all defined in section 3.5, CP/M INTERFACE. They should be used by CALLing location 5 with the relevant value in C for CP/M compatibility. NO guarantee is made of the contents of undefined registers on exit and in general all register contents will be destroyed.

4.8.9 **THE STACK**

The machine code stack pointer is initialised at switch on. It is not recommended that it should be moved as the system can use a considerable amount of stack space. Under no circumstances should routines be written which preclude the use of interrupts; V24 I/O uses them liberally.

4.8.10 **ON/OFF**

The key, as detailed in section 2.4.13, ON/OFF KEY, is software controlled. The keyboard should be scanned periodically in any user written software if manual power OFF is required.

4.8.11 **THE NSC800**

It is not proposed to go into software techniques, meaning of object code, etc., here as the NSC800 is totally software compatible to the Z80. A list of the machine code instructions is presented in section 9.4, NSC800 MACHINE CODE. Users are referred to any tutorial book on the Z80 for detailed programming information.

4.8.12 **EXECUTION TIME**

Of interest in machine code loops, etc., is the execution time of the program. The number of cycles used by each instruction is detailed in section 9.4, NSC800 MACHINE CODE. The cycle time in HUNTER is 250ns.

If serial data is received, then the routines will be considerably slower due to interrupts.

PROGRAMMING TECHNIQUES

4.9 To the programmer, most of HUNTER's features are likely to seem familiar and quite comparable with many other, less portable, microcomputers.

To the operator, HUNTER is likely to be quite different from anything he's encountered before. This fundamental divergence in experience can present a major challenge to the skill of the programmer.

Because HUNTER operators tend to be newcomers to computer techniques, and worse, tend to capitalise on HUNTER's unique physical characteristics by using it far away from the comfort of the computer room, considerable demands are placed on the quality of the programming. These demands are met by making programs as error-free and 'bullet-proof' as possible, together with careful program structures. A HUNTER that interrupts a data entry sequence in the field with "Magnitude Error in Line....." is not likely to be appreciated by the user, or worse, by his customers.

Knowing that field operators would have difficulty in recovering from programming failures on the spot, HUNTER's designers have provided a number of facilities that help overcome these difficulties. But because HUNTER's programming has to have greater integrity than is ever required at the desk-top, there is no substitute for methodical discipline in programming.

The contents of this section explain some techniques that are successfully used to provide reliable and ergonomically friendly user programmes.

4.9.1 DATA CAPTURE TECHNIQUES

A typical HUNTER application program consists of 3 segments:

- A 'Data Capture' segment
- An 'Inspection' segment
- A 'Transmission' segment

All three segments are contained in a common program although treated as independent modules. All share a common database, the 'captured' information, generally stored in array structures.

The operator (as opposed to the programmer) is given a limited range of options within this framework and **never has access to** the Basic interpreter. On power-up, HUNTER will typically

present a menu-style choice of options, often based on the segments or modules themselves.

Once a module is selected, HUNTER will lead the operator through a question and answer sequence until his objective is achieved. Consider the following data capture program:

```
10   REM TELEPHONE NUMBERS
20   REM THIS PROGRAM CAPTURES AND STORES NUMBERS
30   REM UP TO 6 DIGITS LONG.  THEY CAN THEN BE
40   REM RECALLED OR LISTED ON A PRINTER.

100  REM MODULE 0 WHICH FUNCTION?
110  PRINT "PLEASE CHOOSE A FUNCTION, TYPE '1' TO ENTER
     DATA; '2' TO INSPECT OR '3' TO TRANSMIT",
120  INPUT A
130  IF A=1 THEN 200
140  IF A=2 THEN 300
150  IF A=3 THEN 400

200  REM MODULE 1 DATA CAPTURE
210  DIM D(10)
220  FOR N=1 TO 10
230  INPUT D(N)
240  NEXT N
250  GOTO 100

300  REM MODULE 2 INSPECT DATA
310  PRINT "WHICH NUMBER DO YOU WANT TO INSPECT?"
320  PRINT "PLEASE ENTER A NUMBER 1-10!",
330  INPUT N
340  PRINT D(N)
350  GOTO 100
400  REM TRANSMIT THE DATA
410  FOR N=1 TO 10
420  LPRINT D(N)
430  NEXT N
440  GOTO 100
```

This very simple program demonstrates the basic features of much more complex data capture routines, but has one dramatic failing : it is not "bullet proof".

If the operator does not stick precisely to the sequence laid down, the program will soon encounter a problem and resort to error messages that will not help a non-programmer. Because of this, the great majority of effort in programming HUNTER applications is devoted to preventing occurances that might cause confusion.

HUNTER's Basic interpreter contains many features designed to prevent the inexplicable happening in the field: but complex programs can sometimes outwit even their own authors!

A simple solution to this problem is simply to add three more lines:

```
 50  ONERROR 500
500  PRINT "THAT CAN'T BE RIGHT - PLEASE TRY AGAIN"
510  GOTO 100
```

Now, all the error messages from the interpreter are intercepted and re-directed. The program simply tries again. This is fine, but the problem remains: the errors shouldn't be there in the first place!

4.9.2 DATA STORAGE ARRAYS

4.9.2.1 Types of Arrays

Captured data is generally held in array structures created within HUNTER's very large memory. There are three types of array:

- Simple Variables
- Double Precision
- Strings

Simple Variables are used for storage of decimal values of up to 6 digit accuracy.

Double Precision arrays store numbers up to 14 digit accuracy, but use more memory.

NOTE: Both these types store **much larger** numbers than indicated, but only by truncating the least significant digits. For example, the number 12345678 entered as a simple variable would be stored and reproduced as 12345600.

String Arrays store every character, whether numeric or otherwise, literally. They can have any number of characters (up to a maximum of 255 characters in each string). Facilities are provided for converting strings to numbers and vice versa.

4.9.2.2 Array Structures

Every array is denoted by a **name** and placed in memory in a sequential table. Multiple arrays are packed in memory by Basic and accessed via a **Symbol Table.**

Individual array elements are identified by the array name and the element number, together with a character to identify the array type. For instance:

NOTE: Arrays start from element 0

A(10) is the eleventh element of simple variable array A.

A!(2) is the third element of double precision array A! (quite different and independent from array A in the example above).

A$(0)is the first element in string array A$. A$(0) is identical with A$, which can be used for shorthand, useful if there are multiple references to a single string in the program.

In each case, the number in parentheses is the **element number**, and can be variable: A$(A)

or an expression: A$(A*5)

or another array element: A$((6))

Remember the element numbers start from 0!

4.9.2.3 **Creating Arrays**

Arrays can only be created once: if a subsequent attempt is made to re-define an existing array, an error occurs. This is to warn the programmer that he may be accidentally duplicating an array name.

If the statement DIM A$(1,6) is repeated **at anytime** an error message:

```
*DIM Error
```

will appear.

This can be easily avoided by incorporating the DIM in a program sequence that is only executed once, and subsequently branched around. This is often called an **Initialisation Sequence.** A typical format is:

```
10  IF Z<>0 THEN 100
20  DIM A$(1,6)
30  Z=55
```


The variable Z is a flag, indicating that the program has been run before. (All variables are cleared to 0 by CLEAR or any change to program content, but not by power off or removing batteries).

A slightly more sophisticated version is:

```
 10  IF Z<>0 THEN 100
 20  DIM A$(1,6)
100  Z=Z+1
```

Where Z acts both as a flag and as a counter of the number of times the program has been run, a very useful statistic!

4.9.2.4 **Array Sizes**

Arrays are limited in size only by HUNTER's memory space. Remember that arrays compete with program for space, automatically allocated by HUNTER's operating system.

Array size allocation is simplified by **self-sizing.** For this purpose, the Function FRE can be used as a variable. FRE provides the number of memory bytes remaining as follows:

```
After NEW : The total memory available
After program entry : The memory available for data
After array definition : The memory remaining
```

The number FRE can be used to automatically calculate the maximum number of array elements possible for any given array type and to sign on informing the operator of this information.

```
 10  IF Z<>0 THEN 100
 20  N=2
 30  Y=INT (((FRE(0)-7-N*7)/5)-1)
 40  DIM A(Y)
100  PRINT "TOTAL NUMBER OF ELEMENTS AVAILABLE:",Y
```

Note that Y is taken as an integer (INT), that the symbol table entry (-7) is subtracted, that the element size is 5 bytes and that arrays start from zero, so the maximum number is one less (-1). N*7 is an allowance for the number of variables used in the subsequent program, assuming 7 bytes per variable.

NOTE: arrays can also be copied to files. See section 4.14.1

A string array can be sized as follows:

```
 10  IF Z<>0 THEN 100
 20  N=2
 30  Y=INT (((FRE(0)-7-N*7-256)/15)-1)
 40  DIM A$(Y,4)
100  PRINT "TOTAL NUMBER OF STRINGS AVAILABLE:",Y
```

4.9.2.5 **Accessing Array Elements**

Each element in a data array is accessible by its **array index**, expressed in parentheses, e.g:

```
A$(235)
```

is the 236th element in A$

NOTE: Array indexes **must be** within the range declared in the corresponding DIM statement. HUNTER Basic warns the programmer of any attempt to access an element outside the declared range with:

```
*MAG Error
```

Be especially careful with incrementing indexes like FOR NEXT loops.

Array indexes often have a direct relationship to external variables, like stock codes. It is always preferable to **directly** access an array in this fashion in the interests of speed, rather than working sequentially through every element.

If there are discontinuities (gaps) in the existing codes, or if the numbers are too large, a conversion table or algorithm may be needed. Since a literal table (one entry for every code) is likely to be very inefficient, it is generally worth putting considerable effort into deriving an efficient 'tree' structure for direct array access.

4.9.2.6 **Array Searches**

There is often a need to search an array for an element of known content. In Basic, this procedure can be painfully slow.

One solution to this dilemma is to use a specialised machine-code subroutine to search the array and return with the index of the target element.

A much more practical method is the Basic function SRCH.

4.9.3 **DATA INPUT TECHNIQUES**

We are all familiar with the computer truism:

"Garbage in - garbage out"

This was never more true than in data capture applications where out-of-range entries can cause fatal errors that are just embarrassing in the office, but potentially disastrous in the field.

To help avoid the most common form of data entry error, HUNTER provides a unique facility **INPUT USING.** Input using pre-filters incoming data against a predetermined mask and rejects entries that do not fit at the keyboard, warning the operator to try again.

Equally important to the user is optimum use of HUNTER's screen to provide clear, unambiguous prompt messages and input fields.

4.9.4 **USING HUNTER'S SCREEN**

HUNTER's large, 40 x 8 character, LCD screen can be used to positive effect to help its operator. Because the screen is entirely flexible, these notes are provided as a source of ideas rather than as instructions.

This section refers to character-by-character use, more detailed information on direct dot addressing is available in Section 4.7, HUNTER GRAPHICS.

4.9.4.1 **The Virtual Screen**

HUNTER's 40 x 8 character, LCD screen acts as a window to a much larger virtual screen. When characters are written to the screen from a Basic program they are always written to the virtual screen. The commands available in Basic to move the window are 'CLS', 'INPUT' and 'LOCATE'.

'CLS' Clears the virtual screen and sets the window to the top left of the virtual screen.

'INPUT' The execution of an 'INPUT' command will move the LCD window to the part of the virtual screen containing the 'INPUT' prompt.

'LOCATE' The Locate command will re-position the cursor in the virtual screen and will move the window so that the cursor remains visible.

4.9.4.2 **Cursor Addressing**

The cursor position can be commanded from a Basic program, allowing characters to appear anywhere on the screen. Cursor addressing is relative to the virtual screen, therefore characters may not appear on the LCD screen in text mode.

The standard Basic format for cursor addressing is:

```
LOCATE X,Y
```

X and Y are the co-ordinate values expressed in decimal.

4.9.4.5 **The Standard Header Page**

All HUNTER programmes generated at HUSKY Computers feature a standard header page. The header page supplies vital information about the program and use of this format is strongly advocated to other users.

The appearance of a suitable user header page would be:-

```
+-----------------------------------------+
| 20 Oct 1983                   11:20:32  |
|              OPTIONS                    |
|         1 = ENTER  DATA                 |
|         2 = REVIEW DATA                 |
|         3 = AMEND  DATA                 |
|         4 = TRANSMIT DATA               |
|                                         |
| PLEASE ENTER OPTION NUMBER ?            |
+-----------------------------------------+
```

4.9.4.6 **Dynamic Screens**

A vital ergonomic factor in HUNTER program design is reassurance. Many, perhaps most, HUNTER operators have never worked with any kind of computer before and are likely to have some suspicion about this example of 'advanced technology'.

The crucial point is that HUNTER MUST COMMUNICATE WITH THE OPERATOR. If it doesn't he'll soon become frustrated at some situation he doesn't understand and resort to 'traditional' solutions - gently encouraging it with a series of sharp blows or worse!

4.9.5 **HELP STATEMENT**

The HELP facility provides a means of breaking into normal programs to display text for operator assistance.

After the HELP text has been read control is returned to the main program, as if nothing had happened, by pressing the 'HELP' key again.

4.9.5.1 **The Help Vector**

The text displayed by HELP can be linked to the operation currently being performed. The start position of the text can be set under program control. This is achieved by the pair of ram locations 'VECTOR', indicating the memory address of the start of text. This is controlled by the HELP function in Basic.

VECTOR can also be set by POKE.

4.9.5.2 **Text Storage**

Help text is stored by means of REM statements, which allow storing of ASCII text within programs.

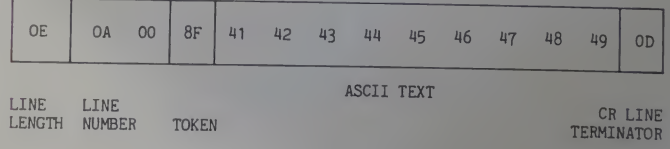
The address in the Help vector points to the start of the line number. The Help program will skip forward over the first four bytes, checking the REM token and then displaying the text.

NOTE: The REM token is stored as 143 Decimal, (8FHex).

Fig 4.2 TEXT STORAGE, shows how the following line of Basic source code is stored:

```
10 REM ABCDEFGHI
```

Fig 4.2 TEXT STORAGE



4.9.5.3 HELP Text Display

The software which controls HELP will scroll forward through the lines of Basic text, or pseudo-Basic text, until the verb found is not a REM. Scrolling backwards is also allowed until a line is found without a REM. In each case, further scrolling is not allowed.

4.9.5.4 Storing the Current Display

The contents and cursor position of the screen are not lost while in HELP mode. On exit from HELP mode, the screen and cursor are restored to their states prior to HELP mode entry.

4.9.5.5 HUNTER Action during HELP

When a program is interrupted for HELP display all normal action is suspended. Serial data reception will continue up to the capacity of the receive buffer. If handshaking is enabled then it will continue in a transparent fashion.

Programs will continue after the HELP key is pressed again.

HELP will be entered during any scan of the keyboard, whether for status or waiting for an actual key depression.

4.9.5.6 Graphics Mode

Because HUNTER's graphics screen cannot be re-created from RAM memory, HELP text would destroy the contents of the screen if allowed during graphics mode. For this reason, HELP is inhibited in graphics mode and help key depressions are ignored.

POWER WARNING

4.10 When HUNTER's batteries become exhausted a power warning message will appear on the screen. Normal operation will continue, but keyboard entries will be punctuated by warning tones and repetition of the power warning message until the battery state is rectified either by replacement of primary cells or recharging of Nickel-Cadmium cells, if installed.

Eventually, if the warnings are ignored, HUNTER will shut down and refuse to operate further.

Every HUNTER has to pass a stringent operating test in this low power regime. However, it is strongly recommended that operator training procedures and user programs are structured to avoid continued operation when warnings become persistent, especially when rechargable cells are used.

The power warning sequence may be misleading if RS-232 communications are used infrequently. When HUNTER's communications system is activated power drain increases significantly. This increase can cause the power warning stage to be missed altogether, and instead cause HUNTER to shut down without warning.

While this is not hazardous, it can be very confusing for an operator who does not know what is wrong. Because of this it is recommended that user programs with infrequent communication requirements use a test routine. This activates the communications package, and then tests the power state before restoring HUNTER to normal. This routine is used at the start of every data entry sequence.

If batteries become exhausted whilst communications are in progress, then normal power warning messages will appear once the keyboard operation is resumed.

NOTE: Power warning messages only occur when keyboard entries are in progress.

SAMPLE WARNING SUBROUTINE

```
20   OUT 132,1
30   J=INP(2):OUT132,0
40   IF J AND 4 = 0 THEN RETURN
50   CLS
60   PRINT "PLEASE CHANGE MY BATTERIES!"
70   BEEP:BEEP:BEEP:BEEP
80   RETURN
```

This program operates as follows:

1) The power status is available as bit 2 in HUNTER port 2. 0 = power OK, 1 = Low Power state.

2) Line 20 energises the RS-232 serial interface.

3) Line 30 reads the power state from port 2, and then closes down the interface to conserve power.

4) Line 40 checks the power status bit.

5) Line 50-80 Display a warning and ring HUNTER's bell!

OFF-LINE PROGRAM STORAGE

4.11 Most HUNTER applications require storage of users' Basic source programs in an easily accessible bulk-memory system for support, updating, exchange and maintenance.

One very popular method of storing HUNTER programs is in user's multi-access mainframe database systems, although minis, micros and other HUNTERs are also used extensively. HUNTER's ability to communicate freely with these other systems is vital to supporting its programming and application.

This section deals with both manual and programmed source code loading and unloading using Basic's complementary LLIST and LLOAD commands.

4.11.1 LLOAD

LLOAD is the command for loading Basic source programs into HUNTER. It may be used to load entire programs or to modify existing ones by appending or overwriting lines.

LLOAD can be used with any of the communications protocols detailed in Part 6, COMMUNICATIONS. During LLOAD no text is echoed to the screen, to speed up the facility. Certain speed/protocol limitations are detailed below.

4.11.1.1 Manual Control

To enter program lines, simply type 'LLOAD' followed by carriage return. HUNTER will then load to memory source text presented on the serial port. Note that syntax is checked on a line by-line basis and that if an error is detected, the LLOAD will terminate and a 'SYX Error' will be displayed on the screen.

If the program provided overflows the memory space available, a 'MEM Error' will appear.

LLOAD mode can be terminated by pressing 'ESC' on HUNTER's keyboard or 'power off'. (Data will not be lost).

Alternatively, an 'ESC' character sent over the interface will return control to the keyboard.

4.11.1.2 **Programmed Control**

Logically, it is difficult for a program to append to or modify itself, since the sequence controlling the modification may be modified as well! However, HUNTER programs can contain sequences that will command a remote processor to send program text and then accept that text as Basic source.

Current versions of HUNTER Basic do **not** support LLOAD as a program statement, e.g. 100 LLOAD. However, the 'Logical Keyboard' Flag has the same effect as LLOAD and can be commanded in a program as a 'POKE' instruction. By setting the flag to the serial input port and then returning to interpreter mode, HUNTER is configured to accept new program lines. These lines may overwrite the lines originally used to enter the LLOAD mode.

Loading may be terminated by sending an unnumbered 'RUN' statement which will cause execution to re-commence from the start of the new or updated Basic program, making the load sequence virtually transparent to the operator.

A possible program format might be:

```
100 REM Program reload routine
110 INPUT "Please enter new program name", X3$
120 LPRINT "Hi there, mainframe.  Please transmit file name",
130 LPRINT X3$
140 POKE  LK,1  :  REM This sets the logical keyboard as the
                   serial input port
150 END : REM Returns control to the interpreter
 .
 .
.The new program lines now load
 .
 .
```

. The last line transmitted is unnumbered RUN, which executes:

```
10 POKE LK,128 : REM Restore the logical keyboard
20 PRINT "New program loaded OK : Continue?"
```

Variable LK is the address of IPFLAG, addresses of memory locations may be found in section 9.7, MEMORY LOCATIONS.

HUNTER's input buffer ensures that the first line of program is captured, no matter how quickly the distant computer responds.

4.11.1.3 **IMPORTANT NOTE**

In present versions of HUNTER operating system, any alteration to program content, including deletion of lines, **automatically** clears all variables and arrays. This is essential to HUNTER's operation, but means that data **cannot** be carried across reload boundaries. Data copied to files is not affected, however.

4.11.2 **LLIST**

LLIST is the principal method of copying and recording HUNTER programs.

Invoking LLIST causes program lines to list sequentially from the lowest line number or from a line number specified as an argument in the LLIST. Examples are :

```
LLIST  Lists an entire program
LLIST 170  Lists from line 170
LLIST 2133  Lists from 2133 or, if 2133 does not exist, from the
next highest line.
```

LLIST terminates when all line numbers are listed, when 'ESC' is pressed or when HUNTER is powered down. Data will not be lost if HUNTER powers down during a listing.

LLIST cannot be incorporated in a program, and is only available for manual use.

Protocol and format selections made in HUNTER's communication package are observed transparently by LLIST. Carriage Return terminator characters are provided at the end of each program line, line feed and null characters are optional.

IMPORTANT NOTE

Remember that HUNTER's communication system needs to power up to its RS-232 state for transmission to occur. This power-up will cause an off-to-low transition that many systems will read as a single, spurious, character. This event only occurs once and can be avoided by powering up the interface prior to activating a receiving device. Methods of pre-powering the interface include :

```
LPRINT " " : REM the interface powers up automatically
OUT 132,1  : REM Direct control of the communications inverter
```

Remember that once powered up, the RS-232 output remains active until HUNTER is powered down or the inverter is commanded 'off' by :

```
OUT 132,0
```

HUNTER power consumption increases substantially during RS-232 transmission with corresponding reduction in battery life.

It should be noted, however, that if the RS-232 line is powered off further serial transmission will be lost unless the interface is powered up again, by:

```
OUT 132,1
```

NOTE: The interface will not power-up automatically with a simple LPRINT statement with no argument.

4.11.3 SPEED/PROTOCOL LIMITATIONS

It is always advisable that some form of protocol handling is established between HUNTER and associated computers.

Some multi-access mainframes and some popular microcomputers are unable to support any form of protocol, however.

In these cases, HUNTER will generally support direct program transfer at speeds of 1200 baud or less. At higher speeds, HUNTER's input buffer may overflow leading to loss or truncation of lines. At 1200 baud, the buffer smooths the flow of incoming program through HUNTER Basic's syntax checking routines.

Problems with truncation will invariably lead to the load operation being aborted and a 'SYX Error' message being displayed, since the Interpreter will not accept partial lines.

The capability of supporting simple 1200 baud communication is invaluable in many situations.

4.11.4 COMMUNICATIONS WITH DATABASES

A very useful HUNTER feature is the ability to communicate directly with mainframe databases using TERM.

This interactive mode can be used manually to establish logon procedures, passwords, etc., before attempting to transfer or copy files or inspect data.

Sequences proven out manually can in most cases be implemented in HUNTER Basic as automatic features of the user's program once

protocols, etc, have been finalised. Such sequences can present inputs to the mainframe and inspect the reply for keywords like 'READY' or simple cursor prompts.

NOTE: Many mainframes are unpredictable in the response they provide to login sequences, with variable 'welcome' or 'news' messages. Make sure your automatic sequence is robust enough to handle these eventualities!

Some general points about database communications should be noted:

4.11.4.1 Rate

Communication occurs generally at 300 baud, although other configurations are possible. Remember that the screen character generator will display all ASCII characters, even if they are not shown on the keyboard.

Other rates encountered on dial-up systems are 110 (very rare) and 1200, but only with sophisticated modems.

4.11.4.2 Parity

Dial-up systems can expect any of even, odd or no parity selections. Transmit and receive parities are generally the same. Always use receive parity if it is available - the occasional appearance of the parity error symbol is a useful indication of bad lines. Remember that telephone lines can be bad in **one direction only.**

4.11.4.3 Full/Half Duplex

Database systems vary widely in the use of full or half duplex operation.

Full duplex occurs when the host system 'echoes' every received character back to HUNTER. In this mode, HUNTER's transmission echo should be selected 'off' so that characters typed on the keyboard only appear on the screen if they have completed the whole circuit of HUNTER - Mainframe - HUNTER.

Basic routines can check full duplex replies as an absolutely secure communication protocol, provided the mainframes' own messages don't confuse the issue.

If the echo switch is left 'on', double characters will appear on the screen. This in no way affects communication, but makes outgoing messages hard to read!

Half duplex occurs when the host does not echo characters back to HUNTER. In this situation, the echo switch should be 'on' to allow outgoing messages to be read by the operator. This method does not guarantee that data sent to the host is being correctly interpreted.

A third mode, **Simplex**, occurs when data is sent in only one direction at one time and is otherwise similar to Half duplex. HUNTER is not concerned which mode is in use.

4.11.4.4 Protocol

Unfortunately, very few dial-up databases support any kind of protocol, so that generally HUNTER's 'none' option should be used. Note that 1200 baud is the fastest recommended speed for no protocol program loading.

However, some systems do support 'XON/XOFF' although implementations vary between computers - seek advice from HUSKY Computers if difficulty is encountered.

4.11.4.5 Other Parameters

The setting of 'NULL' and 'LF' are generally not material in mainframe communication, so that '0' is recommended for NULL count. See below for LF.

4.11.4.6 Terminator

HUNTER's 'ENTER' key generates CR (Carriage Return), but some mainframes expect other terminators. Examples are Control C and LF (Line Feed), or 'New Line'. Consult your computer manual for details.

4.11.4.7 Delete

HUNTER's DEL (Delete) key generates 'rubout'. Few mainframes recognise this, or in many cases allow any deletion at all! 'Cursor left' may work in some cases.

4.11.5 ACOUSTIC COUPLERS

Good quality communication can often be achieved over surprising distances with acoustic couplers at 300 baud, but this method is not recommended for routine daily use.

4.11.6 EXTRA DOCUMENTATION LINES

It is worth knowing that source files kept on computer systems can have further documentation within them by using unnumbered 'REM' lines, e.g.

```
100  PRINT "HELLO"
REM This line of text will be ignored by HUNTER only line
     100 will load into HUNTER.
```

This prevents loading unnecessary text. Naturally, these lines can only be created on the host or database computer.

AUTO POWER FEATURE

4.12.1 Basic supports several powerful functions associated with the POWER key. Basic programs can control precisely what occurs on switch on, or even not allow HUNTER to be turned off. The following commands and statements are available:

ON POWER GOTO
ON POWER RESUME
POWER CONT
POWER n
POWER OFF
POWER OFF RESUME

4.12.2 **DEFAULT CONDITIONS**

The power key behaves as normal - at any stage it can be used to turn HUNTER on or off. The current execution position is lost on turning the power off, and turning it back on returns to the File Manager. However the current Basic program and variables are retained.

4.12.3 **POWER CONT**

a) The power key is disabled.
b) The automatic timeout power down (if no key is pressed within a set period of time) is disabled.

4.12.4 **POWER n**

a) This command overrides a previous POWER OFF.
b) n refers to the automatic power down timeout:

1) n must be in the range 10-255 or 0
2) n=0 is no automatic timeout
3) n is in mutiples of 5 seconds

4.12.5 **POWER OFF**

a) Turns the power off.
b) When power is turned on, Basic program execution starts at the beginning.

4.12.6 **POWER OFF RESUME**

As POWER OFF, but turning HUNTER on continues execution where it left off.

4.12.7 **ON POWER GOTO**

When the power button is hit (during program execution) program execution goes to the specified line number.

4.12.8 **ON POWER RESUME**

a) Powering down proceeds as normal.
b) Powering up begins program execution where it left off.

4.12.9 **Timeouts**

HUNTER has an automatic timeout (see section 2.8.8). This can be modified using POWERn. If HUNTER has been switched off, the power down timeout is set back to the default of 5 minutes unless CONT (see section 3.4.3.4) or Power OFF RESUME (see section 4.12.6) has been used.

FILE HANDLING

4.13.1 HUNTER's Basic interpreter provides the user with the necessary commands and statements to communicate with the File Manager as follows:

CLOSE	MAXFILES
EOF	NAME
INPUT#	OPEN
KILL	PRINT#
LOC(n)	WRITE#

HUNTER's File Manager supports the use of sequential access files. Files are stored in HUNTER's RAM memory, but HUNTER's Basic interpreter will treat them as if they were disk files for reading and writing.

4.13.2 **File Numbers**

When files are OPENed for input or output they are assigned a 'file number' which is specified by the 'AS' parameter. The 'file number' cannot be greater than the value specified in MAXFILES. If no MAXFILES have been specified a default value of one is taken. A file is always referred to by its 'file number'.

4.13.3.1 **File Handling Commands**

A brief description of the file handling commands and statements follow, further details may be found in the relevant section of Part 5, Basic Functions.

4.13.3.2 **CLOSE#**

When files have been OPENed for reading or writing and no further access is required CLOSE will write the end of file marker (EOF) to the file.

4.13.3.3 **EOF**

When reading data from a file a check must always be made after every INPUT# statement for end of file, an attempt to read past the end of file will produce an error.

4.13.3.4 **INPUT#**

This command is used for reading data from a file.

4.13.3.5 **KILL**

Used for deleting files from within a Basic program. The file must first be CLOSEd if it has previously been OPENed for reading or writing.

4.13.3.6 **LOC(n)**

This will return the number of records that have been read or written to a file. The parameter (n) refers to the 'file number' associated with the file.

4.13.3.7 **MAXFILES**

Sets the maximum number of files that may be opened simultaneously; if it is not specified a default value of one is taken. MAXFILES will perform a CLEAR operation when it is executed, destroying all symbol table entries. Therefore it must be defined at an early point in the program.

4.13.3.7 **NAME**

This will re-name a file from within a Basic program.

4.13.3.8 **OPEN**

Before a file can be read or written to it must first be OPENed. The OPEN command has as one of its parameters 'FOR' which has to be specified with one of three options which are:

OUTPUT = Opens and clears file ready for output.
APPEND = Opens a file for output. Records written are added to the end of the file.
INPUT = Opens a file for reading.

4.13.3.9 **PRINT#**

Data is output to the file. Each variable must be delimited by a comma.

4.13.3.10 **WRITE#**

Data is output to the file, variables do not need to be delimited by commas.

ERRORS AND WARNINGS

4.14 HUNTER's Basic Interpreter will display one of a number of messages on an error occurring.

The messages have three characters, generally with a line number. The meaning may be found with reference to the following Table 4.14. They are aimed at experienced programmers.

Each error has an error number. This is the value returned by the ERR verb and is useful during error trapping.

TABLE 4.14

MESSAGE	NO.	MEANING
ARG	0	**ARGUMENT ERROR:** A function has been called with an out of range argument.
STX	1	**SYNTAX ERROR:** Incorrect syntax in a Basic statement.
CSK	2	**CONTROL STACK ERROR:** Incorrect FOR...NEXT; WHILE...WEND or GOSUB...RETURN construction.
SSK	3	**SYSTEM STACK ERROR:** Attempt to POP a non-existant variable.
DIN	4	**DIRECT INPUT ERROR:** Use of a meaningless Basic statement as a command, e.g. NEXT.
DIM	5	**DIMENSION ERROR:** Attempt to redefine an existing variable.
FP	6	**FLOATING POINT ERROR:** Illegal maths operation, e.g. B=A/C where C=0.
LNo	7	**LINE NUMBER ERROR:** Reference to a non-existant line number.
SQR	8	**NEGATIVE SQUARE ROOT:**
MAG	9	**MAGNITUDE ERROR:** Reference to an array element larger than the array dimension.
RD	10	**READ ERROR:** The data statement is either invalid or non-existant.
MEM	11	**STORAGE OVERFLOW:** Variable storage or array overflowed available memory.
ARY	12	**ARRAY ERROR:** Use of an element in an un-defined array.
BAS	13	**BASIC SOURCE ERROR:** Attempt to run a non-Basic program.
HLP	14	**HELP SOURCE ERROR:** The HELP statement has an illegal line number reference.

TABLE 4.14 Contd..

MESSAGE	NO.	MEANING
TYP	15	**TYPE MISMATCH ERROR:** Received string data type instead of numeric.
STR	16	**STRING COMPLEXITY ERROR:** Insufficient space to manipulate a string expression.
RES	17	**RESUME WITHOUT ERROR:**
FNF	18	**FILE NOT FOUND:** Reference to a non-existant file.
IFN	19	**ILLEGAL FILE NAME:** Filename does not conform to standards of file names.
FOP	20	**FILE ALREADY OPEN:**
FNO	21	**FILE NUMBER ERROR:** File number greater than MAXFILE.
FCL	22	**FILE CLOSED ERROR:**
FOO	23	**FILE OPEN FOR OUTPUT ERROR:**
FOI	24	**FILE OPEN FOR INPUT ERROR:**
WWD	25	**WHILE/WEND ERROR:** WHILE without WEND or without WHILE.
DSK	26	**DISK ACCESS ERROR:**
???	27-255	**USER DEFINED ERROR:**


BASIC FUNCTIONS
CONTENTS
5.1 INDEX TO BASIC FUNCTIONS
5.2/27 BASIC FUNCTIONS

INDEX TO BASIC FUNCTIONS

5.10.3	INCHR	I/O Statement	Returns single character from keyboard
5.10.4	INKEY	I/O Statement	Returns keyboard status
5.10.5	INKEY$	I/O Statement	Returns single character from keyboard if input pending
5.10.6	INP	I/O Statement	Returns value at port address
5.10.7	INPUT	Statement	Returns data input from keyboard
5.10.8	INPUT USING	Statement	Validates data input from keyboard
5.10.9	INPUT#	I/O Statement	Input data from file
5.10.10	INSTR	Function	Returns position of second string in first string
5.10.11	INT	Function	Returns integer part of argument
5.11.1	JSR$	Function	Returns fixed-field string
5.12.1	KEY	Command	Initialises soft keys
5.12.2	KEY(n)	Command	Activates/deactivates soft keys
5.12.3	KILL	Command	Deletes file
5.13.1	LEFT$	Function	Returns left part of string
5.13.2	LEN	Function	Returns lengths of string
5.13.3	LET	Statement	Assigns value of variable
5.13.4	LINCHR	I/O Statement	Returns single character from RS-232
5.13.5	LINE	Statement	Draws straight line
5.13.6	LINPUT	I/O Statement	Returns entry from RS-232
5.13.7	LIST	Command	Lists program at LCD
5.13.8	LLIST	I/O Statement	Lists program at RS-232
5.13.9	LLOAD	Statement	Loads program from RS-232
5.13.10	LN	Function	Returns natural logarithm
5.13.11	LOAD	Command	Loads program from file
5.13.12	LOC	I/O Function	Number of records read/written
5.13.13	LOCATE	Command	Sets cursor position
5.13.14	LOG	Function	Returns logarithm to base 10
5.13.15	LOPCHR	I/O Statement	Sends single character to RS-232
5.13.16	LPRINT	I/O Statement	Outputs to RS-232
5.13.17	LTRON	I/O Statement	Sends trace output to RS-232
5.14.1	MAXFILES	I/O Statement	Maximum no. of files to be opened
5.14.2	MID$	Function	Returns mid portion of string
5.15.1	NAME	Command	Re-names file
5.15.2	NEW	Command	Initialises program space
5.15.3	NEXT	Statement	Concludes FOR...NEXT loop
5.16.1	ON BREAK	I/O Statement	Vectors program on BREAK key
5.16.2	ON COM	I/O Statement	Vectors program on communication
5.16.3	ON COMMS	I/O Statement	Vectors program on COMMS failure
5.16.4	ON ERROR	Statement	Vectors program on Syntax error
5.16.5	ON GOSUB	Statement	Conditional branch to subroutine

5.16.6	ON GOTO	Statement	Conditional branch
5.16.7	ON KEY	I/O Statement	Vectors program on soft keys
5.16.8	ON POWER	I/O Statement	Vectors program on POWER key
5.16.9	ON POWER RESUME	I/O Statement	Restarts program on power up
5.16.10	ON TIME$	Statement	Vectors program on system time
5.16.11	OPCHR	Statement	Outputs 1 or more ASCII characters
5.16.12	OPEN	I/O Statement	Opens file for input/output
5.16.13	OUT	I/O Statement	Outputs to specified port
5.17.1	PEEK	Statement	Returns decimal byte value of memory locations
5.17.2	PI	Statement	Value of PI = 3.14159
5.17.3	POINT	Statement	Returns condition of pixel
5.17.4	POKE	Statement	Sets memory location with decimal value
5.17.5	POP	Statement	Returns value from machine code linkage/stack
5.17.6	POS	Statement	Returns cursor position
5.17.7	POWER	Statement	Specify auto time off
5.17.8	POWER CONT	Command	Disables power off key and time outs
5.17.9	POWER OFF	Command	Switches HUNTER off
5.17.10	POWER OFF RESUME	Command	Switches HUNTER off
5.17.11	PRINT	Statement	Outputs to LCD
5.17.12			
5.17.13	PRINT#	I/O Statement	Output data to file
5.17.14	PSET/PRESET	Statement	Set/re-set pixel
5.17.15	PUSH	Statement	Puts value onto machine code linkage stack
5.19.1	READ	Statement	Returns value from DATA statement
5.19.2	REM	Statement	Enters remarks to Basic text
5.19.3	RESTORE	Statement	Resets Read pointer
5.19.4	RESUME	Statement	Restarts program at specified line
5.19.5	RETURN	Statement	Returns from subroutine
5.19.6	RIGHT$	Function	Returns right portion of string
5.19.7	RND	Function	Produces random number
5.19.8	RUN	Command	Starts a program execution
5.20.1	SAVE	Command	Writes program to file
5.20.2	SCREEN	Command	Changes screen mode
5.20.3	SGN	Function	Returns a value for the sign of its argument
5.20.4	SIN	Function	Returns Sine of argument
5.20.5	SOUND	Command	Generates specified tone
5.20.6	SPACE$	Function	Returns string of spaces

5.20.7	SPC	Function	Prints spaces
5.20.8	SQR	Function	Returns square root of its argument
5.20.9	SRCH	Function	Returns target string array position
5.20.10	STOP	Statement	Terminates program execution
5.20.11	STR$	Function	Returns string equivalent of a numerical argument
5.20.12	STRING$	Function	Returns string of characters
5.20.13	SWAP	Function	Exchange contents of two variables
5.21.1	TAB	I/O Statement	Formats Print output
5.21.2	TAN	Function	Returns tangent of argument
5.21.3	TIME$	Function	Returns current time string
5.21.4	TRON/TROFF	Command	Turns trace on/off
5.23.1	VAL	Function	Returns numeric value of string
5.23.2	VARPTR	Function	Returns address of variable
5.24.1	WAND	Command	Defines wand decode software
5.24.2	WHILE..WEND	Statement	Conditional execution of statements
5.24.3	WINCHR	I/O Statement	Inputs a single character from an optical wand
5.24.4	WINPUT	I/O Function	Inputs string from optical wand
5.24.5	WRITE#	I/O Function	Write data to file

ABS

5.2.1 **Function** **ABS(N)** returns the absolute value of the argument.

Syntax ABS(N) where N can be a variable, number or result of a numeric expression.

Examples
```
Y=ABS(-12.345) returns Y=12.345
Y=ABS(-0.5) returns Y=0.5
Y=ABS(0.5) returns Y=0.5

or if V=-27

Y=ABS(V) returns Y=27
Y=ABS(V-50) returns Y=77
```

Remarks The absolute value returned is always positive.

ARG

5.2.2 **Function** **ARG** is used for passing parameters to machine code subroutines.

Syntax D=ARG(N)

Where D is a Dummy Variable
N is the number to be passed

Examples P=ARG(10) Passes 10
P=ARG(V) Passes V

Remarks ARG loads the Z80 E and C registers.

They are loaded with the higher and lower byte portions of a 16 bit representation of the ARG argument.

For example if V in the example above was 4100 then:

E=16 C=4

NOTE: 16x256+4 = 4100

This function is useful when making CP/M system calls to set the C register before a call to location 5. See section 4.8, MACHINE CODE CALLS.

ASC

5.2.3 **Function** **ASC** returns the decimal ASCII value of the first character of a string variable. For a list of ASCII codes used with HUNTER Basic see section 9.2.

Syntax ASC("STRING")

Examples
```
Y=ASC(X$(3))
PRINT ASC("A")
```

will print 65

```
10 T$="AB"
20 PRINT ASC(T$)
```

will also print 65, since "A" is the first character.

Remarks The string argument must be enclosed in parentheses.

ASC is particularly useful for detecting special characters like control codes, and/or distinguishing them from digits 0-9 or characters A-Z.

ATN

5.2.4 **Function** ATN generates the angle whose tangent is specified by the argument.

Syntax ATN(N)

Examples A=ATN(0.6009)

Sets A equal to 0.541071 Radians, i.e. 31°

or

B=ATN(V)

sets B equal to ARCTAN of variable V

Remarks The resultant angle is specified in radians.

To convert to degrees:

A=ATN(V)*180/PI

in the example above,

PRINT (ATN(0.6009)*360/(2*PI))
returns

31.0011

BEEP

5.3.1 **Function** **BEEP** sounds the internal sound generator.

Syntax BEEP

Example 10 FOR I = 1 TO 100
20 BEEP:NEXT I

Remarks This function is equivalent to OPCHR 7 and produces a short tone from the sound generator. For a more varied sound use SOUND.

CALL

5.4.1 **Function** **CALL** allows a machine code subroutine or an existing routine in HUNTER's housekeeping package to be called from Basic and executed.

Syntax D=CALL(N)

where D is a variable
N is a Call Address

Examples 10 A=CALL(0)
20 B=CALL(V) where V is the address to be called

Switches execution to the program located at the location specified. The call in line 10 causes a restart of the HUNTER's software.

Remarks **WARNING:** Use CALL only as indicated in section 4.8, MACHINE CODE CALLS. Use of unspecified calls will crash HUNTER, resulting in loss of data, etc.

When control has passed back to Basic from a machine code program, then D = the final number stored in the Z80 accumulator when the program returns.

CHAR

5.4.2 **Function** **CHAR** specifies the character set.

Syntax CHAR n[,I] 0<n<4

Examples CHAR 2,I

This will output double width text in inverse video if I>0.

Remarks In graphics mode only there are five different character sets, with CHAR 0 being the default set.

I is the inverse parameter. If it is zero then the characters are normal. If it is non zero then inverse video is displayed.

CHAR 0 produces a small 7 x 5 pixel character set defaulted to after a SCREEN 1 statement.

CHAR 1 is the standard 9 x 7 character set.

CHAR 2 are double width characters.

CHAR 3 are double height characters.

CHAR 4 are double height and double width characters.

N.B. Each time character size is changed, the cursor is re-located to the upper left-hand corner.

NOTE: CHAR has no effect in text mode.

CHR$

5.4.3 **Function** **CHR$** performs the inverse of ASC. It returns the single character string specified by the value of N, which may be an expression.

Syntax CHR$(N)

where the result of N is between 0 and 255.

Examples

```
10 DIM A$(10,10)
20 A$ = CHR$(34):REM (see note below)
30 PRINT "HE SAID,";A$;"HELLO";A$
40 B=7
50 PRINT CHR$(B):REM BEEP
60 PRINT CHR$(1):REM THIS CLEARS THE SCREEN
```

The above will print:

HE SAID, "HELLO"

then bleep, and then clear the screen.

Remarks The expression must have the value 0 to 255. The expression must be enclosed in parentheses.

NOTE: (The character " cannot be embedded in PRINT strings since it acts as a delimiter).

CIRCLE

5.4.4 **Function** **CIRCLE** draws a circle of given radius at the specified co-ordinates.

Syntax CIRCLE [(X,Y,)] ,r[,n] 0<X<239 0<Y<63

Example

```
10 FOR I = 20 TO 220 STEP 40
20 CIRCLE (I,32),20
30 NEXT
```

Will plot six circles across the display.

Remarks The optional parameter n will plot the circle in black if it is odd, or erase the points if it is even. When omitted, it defaults to odd. The co-ordinates may also be omitted, and the circle is then centred on the current graphic cursor position. Circles plotted off the edge of the screen will produce wrap around, Modulo 256.

The CIRCLE command range checks the x-coordinate, y-coordinate and radius to be less than 256. Since the HUNTER screen in graphics mode can only display 0-239 and 0-63 in the x and y directions respectively, there are occasions when the expected circle does not appear on the screen.

CLEAR

5.4.5 **Function** **CLEAR** will set all variables defined in the applications program to zero.

Syntax CLEAR

Examples CLEAR

resets all variables.

Remarks Note that the variables are also set to zero if at any time program lines are altered or added. Variable contents are otherwise maintained indefinitely by HUNTER, as described earlier. Clear also resets the FOR/NEXT loop stack and arithmetic stack.

CLOSE#

5.4.6 **Function** **CLOSE#** will close any file that has been OPENed.

Syntax CLOSE[#file 1,#file 2,.....#file X]

Examples CLOSE#4,#5

Will close file numbers 4 and 5.

Remarks If CLOSE is specified with no parameters all files that have been OPENed are CLOSEd.

When the CLOSE command is executed, HUNTER's firmware automatically performs a checksum generation on the file and stores the value in the file directory for checking the next time the file is opened.

Failure to CLOSE the file in the correct manner will result in a 'no file' error message when an attempt is made to OPEN the file at a later date.

When a file is closed, any data in the buffer not already written to the file will be saved. This means that there must be space available in the HUNTER for a file to be closed correctly. It is therefore recommended that the following method be used to ensure data integrity in all application programs where the above problem may arise.

```
10 OPEN "EXPAN.DTA"FOR OUTPUT AS 1
20 WRITE ##1,A:CLOSE
```

This reserves an area of memory in the disk space. Using ONERROR to trap a "disk full" error, the handling routine could simply be:

```
1000 KILL"EXPAN.DTA"
1010 CLOSE
```

This ensures that there will always be enough room to close all files.

CLS

5.4.7 **Function** **CLS** clears the display screen.

Syntax CLS

Example 100 CLS

This clears the screen ready for text or graphics output.

Remarks In either graphics or text mode, CLS will clear the LCD display. It has a similar effect to OPCHR 1. In graphics mode, the screen may also be cleared (and the character set initialised to CHAR 0) with SCREEN 1.

NOTE: Only the screen mode currently in use (text or graphics) is cleared when CLS is used.

COM

5.4.8 **Function** **COM** activates or deactivates the communications interrupt routine.

Syntax COM ON
COM OFF
COM STOP

These statements must be initialised with an ON COM statement.

Example
```
10 ON COM GOSUB 1000
20 COM ON

1000 REM TRAP ROUTINE
1010 LINPUT A$ ...

1990 RETURN
```

Remarks These statements decide what action is taken on receiving communications. The interrupt occurs on the first character.

COM ON When executed, allows trapping of communications by the ON COM statement. If the previous state was COM OFF then the buffer is cleared by this statement.

COM OFF The trapping is deactivated and all communications interrupts are ignored.

COM STOP Communications interrupts are noted but not acted upon by the ON COM statement until a COM ON is executed.

CONT

5.4.9 **Function** **CONT** continues Basic program after STOP.

Syntax CONT [Line number]

'Line number' is optional and allows the program to continue from a different line number.

Examples CONT continue from next line

or

CONT 1000 continue from line 1000

Remarks Allows a program to continue execution from the line following the occurrence of:

a) detection of the break key
b) execution of a 'STOP' statement
c) execution of an 'END' statement
d) detection of a syntax error

By specifying a line number argument continuation can occur from that line number.

In the case of multiple statement lines, execution will begin from the next line.

COS

5.4.10 **Function** COS returns the Cosine of the argument.

Syntax COS(Angle)

Where angle is in radians.

Examples X = COS(Y) sets X equal to the cosine of Y.
R = COS(5) sets R equal to the cosine of 5 radians.

Remarks **NOTE**: For angles in degrees use:

X=COS(A/180*PI)

CRT

5.4.11 **Function** **CRT** causes HUNTER to treat the serial port as the display and keyboard rather than the internal LCD display and keyboard.

Syntax CRT

Examples CRT

Remarks CRT sets keyboard and display functions to the RS-232 port.

The command is useful for entering programs quickly also characters not on HUNTER's keyboard may be used.

Any standard VDU may be used. Baud rates and other parameters are set on HUNTER using 'Initialise Communications'.

Remember that a 'crossed' RS-232 lead is needed with GND,RXD and TXD circuits installed.

CUROFF/CURON

5.4.12 **Function** **CUROFF/CURON** turns cursor off and on.

Syntax CUROFF
CURON

Example CUROFF

To remove the cursor when display in use.

CURON

This will turn the cursor on.

Remarks HUNTER defalts to CURON following power up.

DATA

5.5.1 **Function** **DATA** statements hold constants for use in subsequent READ statements.

Syntax DATA item 1, item 2....., item n.

Examples
```
100 DATA 12,A1,25,SIN(0.5),MID$(A$,3,4),99
110 DATA 5,6,7,8,6*5,"HELLO"
```

After reading the value 99 the READ will take 5, and so on, until every data value is used. This form of data storage is not affected by CLEAR or program line changes which will erase variables. To return to the first value in a DATA list use the RESTORE statement.

A data statement must appear on a program line by itself. It cannot form part of a multiple statement.

Remark The arguments for DATA statements can be either numerical values, variables, or expressions. Multiple arguments are separated by commas.

String variables may be used in DATA statements. Direct text must be enclosed in quotes.

NOTE: Care must be taken that variables in the corresponding READ statement are of the same type as the DATA.

DATE$

5.5.2 **Function** **DATE$** returns a ten character string including the day, month and year separated by dashes. It may be used both as a statement and a variable.

Syntax A$ = DATE$ or DATE$ = B$

Example
```
10 DATE$ = "5/12/86"
20 PRINT DATE$
```

Remarks The ten character string returned by DATE$ is in the format MM-DD-YYYY. As in the example, when DATE$ is used as a variable leading zeros may be omitted and the year defaulted to 19yy.

DATE$ will accept /or : or - delimiters to set the date, but will always print the date using -.

DAY$

5.5.3 **Function** **DAY$** returns a three character string, it may also be used for setting the correct day.

Syntax DAY$ [="MON"]

Examples DAY$="MON"

will set DAY$ to MON

100 PRINT DAY$

will output:

Mon

Remarks The day string may be input in upper or lower case.

Acceptable days are:

MON
TUE
WED
THU
FRI
SAT
SUN

DAY$ always outputs the day with one upper case character and two lower case.

DEFSEG

5.5.4 **Function** **DEFSEG** defines the page of RAM addressed by subsequent CALL, PEEK or POKE commands.

Syntax DEFSEG = page no.

Example DEFSEG=1

sets RAM page to 1

DEFSEG may also be used as part of an expression

PRINT DEFSEG*3

Remarks Page no. is an expression from 0 to the last page in the system. When HUNTER is powered up, DEFSEG is set to 0. This is the TPA or workspace, most often used for machine code programs called from Basic.

After a program error, DEFSEG is set back to 0.

This page is intentionally left blank.

DIM

5.5.6 **Function** DIM allocates arrays within HUNTER.

Syntax DIM variable name (array size)

DIM double precision variable name (array size)

DIM string variable name (array size, element size)

where array size and element size are numeric expressions.

Examples DIM A (25)

defines an array variable A of 26 elements, including the element A(0). The array size must be positive.

DIM J4(100),J5(100),Q(1000),J(N)

defines multiple arrays. N is a previously defined variable.

Remarks There are three kinds of DIM statement in HUNTER Basic : Simple variables, double precision variables and strings.

NOTE: The maximum number of elements in a single array is 16,384.

All array types have a range from 0 (zero) to the limit defined in the DIM statement.

Arrays can be allocated to any combination of variable names.

Double precision arrays are defined in the same way except for the use of "!". Each element has a 14-digit size.

DIM A!(25)
DIM AZ!(32)

define double precision arrays.

Space for string arrays is reserved in HUNTER memory using

a DIM statement followed by string name, the number of elements in the array, and the maximum string length required.

```
DIM A$(25,10)
```

defines a string array with 25 elements each of maximum length of 10 characters.

Each character in a string requires 1 byte of storage.

Since HUNTER retains variable definitions indefinitely, DIM statements should not be included in normal user program sequences. Instead, arrays should be initialised by a separate routine or defined directly i.e. without line numbers.

A useful technique is to set in the application software a switch indicating whether the program is being used for the first time or not. If the switch is a variable which is made non zero when the program is first run, the fact that any modifications or use of the CLEAR statement will clear the value to zero can be used to determine whether the arrays should be redefined or not.

```
10 If A = 1 THEN GO TO 100
20 DIM J(50)
          .
          .
          .
90 A = 1
100 REM user program starts here.
```

NOTE: That the variable A could be used as a counter to indicate the number of times that the program has been run.

Default values for numeric arrays are 10 elements. Strings default to one element of 20 characters, larger strings have to be dimensioned.

EDIT

5.6.1 **Function** EDIT invokes the Hunter text editor.

Syntax EDIT [filename]

Examples EDIT

The editor is invoked. If a Basic program is currently loaded in the workspace, the editor will operate on it. If no program is loaded, entry of new text or program may proceed.

EDIT "PROG1"

Program PROG1.HBA is first loaded into the workspace. EDIT is then invoked.

Remarks For full description of the editor see section 7, EDITOR.

EDIT cannot be used as a program statement.

To exit EDIT, either save the file (CTL/FN 6, followed by filename) or Exit (CTL/FN 8). The edited version of the program will remain in the workspace.

NOTE: Following 'Save' or 'Exit' commands, the editor will display a message:

** Wait

A few seconds may elapse while the editor re-tokenises the Basic source text into the more compact format of the .HBA file.

Only files with the .HBA suffix are treated in this way. When invoked from Basic, the editor assumes .HBA as a default suffix.

END

5.6.2 **Function** END terminates user program execution.

Syntax END

Examples 1000 END

when executed, displays:

READY

and returns control to the Basic command level.

Remarks END does not require an argument.

This is normally the last line of a program although Basic will default an END even if it does not exist. END may also be used ir the middle of programs for terminating it.

Generally user programs do not END but retain control over HUNTER operation.

EOF

5.6.3 **Function** EOF checks for end of file.

Syntax EOF(file no)

Examples

```
10  OPEN "DATA.TXT" FOR INPUT AS #1
20 INPUT#1,A,B,C
30 PRINT A,B,C
40 IF EOF(1)THEN CLOSE #1 ELSE GOTO 20
50 PRINT"END OF FILE":END
```

Remarks EOF = -1 if end of file has been reached
EOF = 0 if more data is in the file

Programs should not try to read beyond the end of a file since an error will occur.

NOTE: The file DATA.TXT used in this example, could be created by a Basic program of the form:

```
100 OPEN "DATA.TXT"FOR OUTPUT AS #1
110 WRITE#1,123,235,345
120 CLOSE#1
130 END
```

This will ensure that the correct end of file marker is written to the file.

ERR/ERL

5.6.4 **Function** **ERR** and **ERL** return the error code and the line number at which the error occurred respectively.

Syntax X = ERR Y = ERL

Example

```
10 GOTO 50
20 A = ERR
30 PRINT "Error Number";A;"at line";ERL
50 ON ERROR GOTO 20
60 B=A/0
70 STOP
```

This example prints:

Error Number 6 at line 60

due to the floating point error caused by an attempt to divide by zero.

Remarks ERR and ERL will always contain the codes of the last error, and will often be used in conjunction with ERROR.

These functions are often used in communications with the ON ERROR GOTO trap.

NOTE: See section 4.14 for actual error numbers.

ERR defaults to zero until an error occurs.

ERROR

5.6.5 **Function** **ERROR** simulates a Basic error with the advantage that it allows user defined codes. For a list of Bsic error messages, see section 4.14.

Syntax ERROR n

where 0<n<255

Example

```
10 INPUTa,b
20 ONERRORGOTO100
30 IFa>bTHENERROR156
40 GOTO10
100 IFERR = 156THENPRINT"a>b error"
110 RESUME 10
```

Remarks The ERROR statement neatly traps errors and forces the printing of a Basic error message. If the error number is undefined in Basic, then processing can be forced to restart at another line to print a user defined message. When ERROR is set to n, this number may be recovered with ERR, and the line number with ERL.

After a forced user error trap, a suitable RESUME command must be executed to ensure correct program operation.

EXP

5.6.6 **Function** **EXP** generates the value of e raised to the power of X.

Syntax EXP(X)

Examples A!(0)=EXP(3)
(A!(0)=e to the third power)

B=EXP(N)
(B=e to the power of N)

Remarks X must be in the range -290 to +290.

If X is not in this range then a 'MAG Error' will occur.

X may be an expression.

e is defined as: 2.7182818284590

FILES

5.7.1 **Function** **FILES** displays all the current user files in HUNTER.

Syntax FILES

Example This program will list all the current user files in HUNTER:

```
6030 CLS:PRINT"Please enter the file you wish to select:"
6040 LOCATE0,1:PRINT"(Use shift arrow keys to scroll)"
6050 LOCATE0,3:FILES
6060 LOCATE0,2:INPUT"enter filename  ";a$
```

FIX

5.7.2 **Function** FIX reduces the argument to an integer.

Syntax A=FIX(n)

Examples
```
A=FIX(12.37)
sets A = 12

A=FIX(-5.95)
sets A=-5
```

Remarks FIX should not be confused with INT which will round negative numbers.

FOR

5.7.3 **Function** **FOR** executes a series of instructions in a loop a given number of times.

Syntax
```
FOR  N = A TO B STEP C
NEXT N
```

Where N is a variable, A is a numeric start value, B is a numeric end value and C is an increment/decrement value.

A, B and C can be numbers, variables or expressions.

Examples
```
10 FOR A = 1 TO 100 STEP 10
 .
 .
 .
100 NEXT A
```

causes the value of A to equal 1,11,21 etc., for each execution of the loop.

To obtain a **decrementing** count negative values of STEP are used.

```
1000 FOR Z1=91 TO 10 STEP -.1
  .
  .
  .
1050 NEXT Z1
```

Remarks When followed by a NEXT statement, FOR will execute the intervening parameters for the number of times indicated by the values A and B divided by N to the formula:

(B-A+2)/N

When terminating FOR...NEXT loops it is important to end using NEXT and **not** GOTO. Alternatively, exiting via a RETURN when in a subroutine will not cause a build up of the 'control stack'.

Failure to do this will result in a control stack error.

FRE

5.7.4 **Function** FRE returns the number of free bytes left in execution memory (RAM).

Syntax FRE(0)

0 is a dummy argument.

Examples PRINT FRE(0)

displays amount of free space in execution memory.

or

A=FRE(0)

Sets A = amount of free space.

Remarks Note that if a program has not been RUN then the variables will not have been assigned space. This enables easy display of program size and, after RUNing, the total memory usage.

The latter examples allows the AUTO SIZING of arrays to their maximum size.

e.g. DIM A((FRE(0)-10240)/5)

This automizes A leaving 10K (10240 byte) for other variables. "5" is the number of bytes used for each member of A. Remember to leave enough space for all simple variables.

GOSUB

5.8.1 **Function** **GOSUB** causes execution to jump to a Basic subroutine located elsewhere in the applications program.

Syntax GOSUB line number
GOSUB numeric expression

Examples GOSUB 150

causes program execution to jump to a subroutine located at 150.

The argument can also be an argument or expression, for example:

GOSUB SI

or

GOSUB SI+100*A

This is illustrated by the program sequence:

```
10 INPUT A
20 SI=1000
30 GOSUB SI+100*A
40 STOP

1000 PRINT "THIS IS SUBROUTINE 1"
1010 RETURN
1100 PRINT "THIS IS SUBROUTINE 2"
1110 RETURN
etc.
```

If the number 0 is entered in line 10, the program will perform subroutine 1. Input of the number 1 will cause subroutine 2 to be executed.

Remarks A subroutine must always be terminated by a 'RETURN' state ment. Program execution then continues from the statement immediately after the GOSUB. GOSUB always requires a line number as an argument.

GOTO

5.8.2 **Function** **GOTO** branches unconditionally to a specified line number.

Syntax GOTO line number
GOTO numeric expression

Examples GOTO 50

causes program execution to commence immediately at line 50.

10 GOTO 100

causes program execution to skip from line 10 to line 100.

The line number may also be a variable or an expression. Examples:

GOTO A or GOTO A*100

are equally valid.

The program sequence:

```
20 ZB=150
30 GOTO ZB
150 PRINT "THIS IS LINE";ZB
```

would give: THIS IS LINE 150

Remarks GOTO is always used with an argument (line number). If used as an immediate command, GOTO will cause program execution to commence from the line number specified. If used as part of a program, GOTO will cause program execution to skip to the line number specified. If the line specified is not present in memory **HUNTER** will display:

LNo Error in line 150

If the GOTO line was unnumbered, the error message line number will be meaningless.

HELP

5.9.1 **Function** **HELP** displays lines of text on HUNTER's screen when the HELP key is pressed.

Syntax HELP line number
HELP numeric expression

Examples

```
1010  REM  THIS IS A PIECE OF HELP TEXT
1020  REM  INTENDED TO ASSIST THE OPERATOR
1030  REM  IN THE USE OF HUNTER
1040  REM  PLEASE PRESS HELP TO RETURN
1050  REM  TO PROGRAM

10000 HELP 1010
```

Execution of line 10000 will cause the block of remarks to be used as the HELP text.

Remarks HELP requires an argument which must be a valid line number. The referenced line must be a REM statement. When the operator presses HELP, text from the referenced REM is placed on the screen. Use of the cursor up and cursor down keys causes scrolling through a contiguous block of REM statements. The HELP text start may be changed by executing another HELP statement.

Normally, the help key on the HUNTER keyboard is disabled. Execution of HELP causes the HELP code to be loaded into the keyboard table, therefore enabling the use of the key. Once the HELP statement has been executed, it is not possible to disable the HELP function except for deleting the HELP code in the keyboard map.

NOTE: On power up the help key will be again disabled and therefore it is necessary to execute the HELP statement within the program initialisation.

It is recommended that all the HELP text be written contiguously so that the operator may scroll through all of it. Scrolling stops when any non-REM line is encountered, so blocks of text can be easily partitioned if required.

NOTE: HELP text cannot be displayed if HUNTER is in graphics mode and operation of the HELP key is ignored. It is then necessary to use "shift PWR" to turn off HUNTER.

IF

5.10.1 **Function** **IF** selectively executes program statements dependant on result of an expression.

Syntax IF A(op)B THEN C

Where A is an expression, variable or constant and (op) is an operator =, < , > etc. The operator may be logical or omitted. See section 4.2.3, LOGICAL OPERATIONS, for further details.

B is an expression, variable or constant

C is either a line number or program statement.

Examples IF A = B THEN 100
IF A>B THEN PRINT "NO"

C may be a multiple statement which will be entirely executed if the condition is true, but skipped if not.

HUNTER also permits nesting of IF statements.
Example:

IF A= B THEN IF C=D THEN PRINT "FINISHED":GOTO 1000

String expressions may be tested for equality or unequality, for example:

IF A$ + B$ = "ABCDEAB" THEN 100

or

IF A=B AND B=C OR D=5 THEN PRINT"TRUE"

In this example "TRUE" may be printed by one of two methods allowed by the OR operator as follows:

1) if A=B=C
2) if D=5

However,

IF A=B AND B=C AND D=5 THEN PRINT"TRUE"

will only print "TRUE" if all conditions are satisfied.

IF A THEN PRINT"TRUE"

will print "TRUE" only if the variable A is non-zero, i.e. if A=-19 then "TRUE" will be printed.

See section 4.2.2, EXPRESSIONS AND OPERATORS, and 4.2.3, LOGICAL OPERATIONS, for precise details of logical operators.

Remarks For strings of unequal length the equality will be true if the string on the left of the equates is equal to or part of the string on the right as follows:

IF A$ = "ABC" and B$ = "ABCD"

THEN

A$ = B$ is true
B$ = A$ is **not** true

IF..THEN..ELSE

5.10.2 **Function** **IF...THEN...ELSE...** executes either of two statements as the result of an expression.

Syntax IF (expression) THEN (statement1) [ELSE (statement) 2]

Examples IF A=B THEN X=10 ELSE A=A+1

IF D AND NOT B THEN C=0:GOTO50 ELSE RETURN

Multiple statements separated with colons are implemented or skipped depending on the condition.

IF A=6 THEN IF B>10 THEN A=0 ELSE ELSE A=A+1

Omitting the first ELSE will move processing straight onto the next line if the first condition is false (ie A=6) and cause A to be incremented if A=6 and B>10.

Remarks The expression must be able to be evaluated true or false. If the expression is true then statement 1 is executed, and if false statement 2 is executed.

The statement may include other conditional statements (nested), but care must be exercised in the positioning of ELSE to associate them with the correct IF. Generally, each ELSE is associated with the most recent IF. Extra ELSE statements causing no action at the end of the line may be omitted.

The use of logical operators outlined in the section relating to 'IF' are also allowed when using IF...THEN...ELSE.

INCHR

5.10.3 **Function** **INCHR** inputs a single character from the keyboard.

Syntax INCHR variable name
INCHR["PROMT STRING";] variable name

Examples 10 INCHR "DECIMAL VALUE";A
20 PRINT A
30 GOTO 10

displays:

DECIMAL VALUE ?A 65
DECIMAL VALUE ?B 66
etc.

Remarks INCHR does not wait for ENTER to be pressed. A prompt message is displayed as with INPUT. Only one variable may be entered. The variable is set to the decimal value of the character, as detailed in section 9.2, ASCII CHARACTER SET.

A comma following INCHR or the prompt message will suppress the prompt character '?' The comma replaces the semicolon in the syntax.

NOTE: No values are returned for SHIFT, HELP, CONTROL or POWER keys, although these function normally during INCHR.

'Escape' cannot be used to return to Basic from an INCHR loop, since it simply returns the value 27!

INKEY

5.10.4 **Function** **INKEY** checks the keyboard for the operation of a key.

Syntax INKEY A

Examples INKEY A = 65 THEN STOP

detects capital A and stops.

Remarks The variable A is attributed the value of the key pressed. If no key is detected then the variable is equal to zero.

INKEY does not wait for a key to be pressed; execution continues immediately at the next statement.

INKEY is the faster method for obtaining the keyboard status.

INKEY$

5.10.5 **Function** **INKEY$** reads the keyboard and returns a single character if an input is pending on the keyboard.

Syntax A$ = INKEY$

Example

```
10 A$ = INKEY$
20 IF A$ = "" THEN 40
30 PRINT A$
40 GOTO 10
```

Remarks If a key has not been pressed then a null string is returned. The ASCII code of each key, including control codes, is read and assigned to the one character string variable A$.

The characters entered into the program in this manner are not displayed on the screen unless explicitly printed (line 30 in the example). One application may be to insert a pause in a program before displaying another screen of information:

```
80 PRINT"last line of last screen"

90 A$=INKEY$: IF A$="" then 90
100 PRINT"First line of next screen"
```

If a function key is pressed then a two character string is returned consisting of the '^' character and the number of the key pressed i.e. Pressing Function Key 1 will return the string "^1" to an INKEY$ function.

INP

5.10.6 **Function** **INP** inputs a byte from the designated port.

Syntax INP(PORT)

where PORT is a numeric Port Number.

Examples A = INP(132)

Remarks Refer to section 9.8, PORT ALLOCATIONS, for details of addresses and functions.

NOTE: A 'port' is a term used at the hardware level of Hunter. It refers to the input/output addresses used by the NSC800.

INPUT

5.10.7 **Function** **INPUT** obtains a line of data from HUNTER's keyboard

Syntax INPUT[;]["prompt string";] var1, var2, ...var n

Examples INPUT A

Inputs a numeric value from the keyboard and places it into the variable A.

INPUT"Parameters";A,B,C$

Inputs the values of A, B, C$ with the prompt message:

Parameters?

A comma following the prompt string supresses the question mark prompt following the prompt string:

INPUT"Parameters",A,B,C$

will prompt:

Parameters

If no prompt message is required at all then the format entered is:

INPUT"",A

A semicolon prior to the prompt string will inhibit the echo of the users 'Carriage Return'.

Remarks Note that input lines are always terminated by pressing the 'ENTER' key.

INPUT cannot be used as a direct statement.

NOTE: If INPUT's argument is a simple variable and an invalid entry is made, INPUT will simply re-prompt without an error message. An example is:

```
10 INPUT A
```

The user types 'HELLO'. After 'ENTER', HUNTER re-prompts on the next line:

```
? re-do from start
?
```

and awaits a numerical entry.

INPUT USING

5.10.8 **Function** **INPUT USING** obtains a line of data from HUNTER's keyboard and validates this data with a user defined mask.

Syntax INPUT USING (validation string, [min,max]) prompt string, variable name.

INPUT USING (validation string, [min,max,E]) variable name where min, max are numeric expressions and E is a special command.

Examples INPUT USING ("A999",2,4)S$

Causes ? prompt

A minimum of 2 characters must be entered up to a maximum of 4.

The inputs are stored in string expression S$. The first entry must be a letter in the range A-Z, the remaining entries must be numbers in the range 0-9.

INPUT USING ("A999",,4)S$

The same as above, but minimum defaults to 1.

INPUT USING ("A999",2,)S$

A minimum entry of 2 and maximum defaults to 95. Any entries after the fourth are validated against WILD.

INPUT USING ("A999")S$

Minimum and maximum default to 1 and 95 respectively.

INPUT USING (M$,3,6)S$

A minimum of 3 and up to a maximum of 6 entries are allowed. The entries are validated against the string expression M$. The user will receive a '?' prompt.

INPUT USING (M$,3,6)"ENTER",S$

Same as above, but with an 'ENTER' prompt.

INPUT USING (M$,3,6,E)"ENTER",S$

Same as above, but after the 6th character has been entered HUNTER will assume that the enter key has been pressed.

Remarks Entry fields can be made numeric, alphanumeric, etc., in flexible configurations. Entries that do not fit the mask are rejected with a bleep.

INPUT USING is an extension of the standard INPUT statement. However, as each entry is made by the operator, the character is checked against the defined check string. A value which is out of range will cause an audible bleep to occur, rejecting the invalid entry.

The input validation string consists of defined validation characters either in the form of a string expression e.g. A$,B4$ etc., or a direct string enclosed in quotations. Input validation string may **NOT** be omitted.

The optional min and max expressions define the minimum number and maximum number of entries required. If neither min nor max are specified then default values of min 1 and max 95 are assumed. If maximum exceeds the number of characters in the input validation string, then the entries for which there are no validation characters are validated automatically with WILD card characters. If min=0 an input of 'CR' only is allowed.

The values min and max can be variables or arithmetic expressions. Min must not exceed max or a syntax error will occur. Min must not be negative or a magnitude error will occur. Min and max must not exceed 95 or a syntax error will occur.

'Enter' terminates entry. ' Enter' will not be accepted if the number of characters is less than minimum.

When the number of characters exceeds maximum no more characters will be accepted. A warning tone will sound. The cursor right and cursor left are available in INPUT USING fields to edit or correct data prior to pressing 'ENTER'.

The E parameter is optional. If it is specified then the 'Enter' key does not have to be pressed to terminate the end of the characters allowed by the validation mask.

CAUTION: If the input field extends over more than one HUNTER screen line confusing results may occur in subsequent attempts to delete or edit characters.

To suppress the "?" prompt, a comma should follow the prompt string. There must be a prompt string when using comma, therefore, to have no prompt at all it is necessary to use the null string "".

A string, enclosed by quotes, immediately following the validation expression will cause that string to act as the prompt.

The cursor left and right keys are enabled in order to allow the user to modify incorrect entries by simply overtyping them. The delete key will delete the last character entered.

VALIDATION CHARACTER TABLE

A	A-Z only
B	A-Z space
C	A-Z 0-9
D	A-Z 0-9 space
N	0-9 + - . ,
9	0-9 only
.	Decimal point only
*	Wildcard

NOTE 1: Wildcard allows a character in the range of space - del which includes all letters and numbers.

NOTE 2: Control characters are not accepted. Other characters than those in the table will give a Syntax error when the statement is RUN.

INPUT#

5.10.9 **Function** **INPUT#** reads data from file.

Syntax INPUT#file no,var1,var2,....varX

Examples INPUT#1,A,B$,C$

This will input data to the variables A,B$ and C$ from file number 1.

Remarks The variable specified in the INPUT statement should match the type of data stored in the file.

The End of File function EOF should be checked to detect if the end of the data has been reached. See 5.6.3. for examples.

INSTR

5.10.10 **Function** **INSTR** searches for the occurrence of one string in another, and returns with the position of the first match.

Syntax Y = INSTR ([X]A$,B$) 1<X<255

Example
```
10 A$ = "BASICS"
20 PRINT INSTR(A$,"S")
```

This program will print 3, since this is the first occurrence of the letter S.

```
10 A$ = "SUPER BASICS"
20 B$ = "SI"
30 PRINT INSTR(A$,B$)
```

This program will print 9 as the first occurrence of SI.

```
10 A$ = "BASICS"
20 PRINT INSTR (4,A$,"S")
```

The above program will print 6, since this is the first occurrence of the letter S from the fourth character onwards in the string A$.

Remarks If B$ does not occur in A$, then 0 is returned. X, which is optional and defaults to 1, is used to start the search at a particular place within a string.

Case is also important, i.e. "s" is not the same as "S".

INT

5.10.11 **Function** **INT** returns the truncated value of the argument, if positive, or rounds the value if negative.

Syntax INT(N)

Where N is a numeric expresion

Examples A = INT(123.456)

sets A equal to 123.

Remarks **NOTE:** Negative numbers will be rounded in a negative direction.

A=INT(-4.75)
Sets A equal to -5

FIX is an alternative method.

JSR$

5.11.1 **Function** **JSR$** creates a string of given length where the numeric argument is right justified and padded with leading zeroes.

Syntax JSR$(A,N)

A=Numeric argument
N=length

Examples A = -12.34

JSR$(A,8) returns -0012.34
JSR$(A,5) returns -12.3

A = 12.34

JSR$(A,8) returns 00012.34
JSR$(A,5) returns 012.3

Remarks For positive numbers the '+' signs are replaced by '0'.

Sign (0 and -) and decimal point each count as one character.

KEY

5.12.1 **Function** **KEY** enables/disables display of, and redefines soft key values.

Syntax KEY ON

KEY OFF

KEY LIST

KEYn,string 0<=n<=8

Example KEY ON

Allows the soft key display to be displayed on the bottom line of the display. This line is then not scrolled.

KEY OFF

Erases the soft key display from the bottom line of the screen, allowing this line to be used by the user. It does not inhibit the use of soft keys.

KEY LIST

This displays the entire string assignments of the eight soft keys on the screen.

KEYn,string

The keys are programmed by writing a string, of which the first four characters are displayed on the bottom line of the screen if the soft key display has been enabled.

KEY1,"?X,Y"

will set key 1 to print the values of the variables x and y when struck. Using the soft keys is equivalent to using the keyboard - but faster.

KEY 1, "?TIME$"+CHR$(13)

will display the time when pressing soft key 1.

REMARKS On entering Basic, the soft keys default to:

KEY 1: File(s)	KEY 5: Save(")
2: Run	6: Load(")
3: Edit	7: Kill(")
4: List	8: Syst(em)

However, if the auto-run mode on power up has been selected, the soft keys are left as they were last programmed.

When using the KEYn,string command, if KEY0 is typed the soft keys revert to their power up default values. If n is not in the required range then a 'Magnitude Error' occurs.

If it is required to terminate the string with a carriage return or Enter, then it is necessary to terminate with "string" +CHR$(13) as control characters are not accepted in command lines.

KEY(n)

5.12.2 **Function** **KEY(n)** activates and terminates trapping of soft keys.

Syntax

```
KEY(n) ON
KEY(n) OFF
KEY(n) STOP
```

n is a numeric expression in the range 1 to 8 to reference the soft key.

Example

```
10 ON KEY (1) GOSUB 1000
20 KEY (1) ON
.
.
.
900 END 'FINISH OF MAIN ROUTINE
1000 REM SOFT KEY (1) TRAP ROUTINE
1990 RETURN
.
.
.
```

Remarks The KEY(n) statement is used for trapping either soft key entries or cursor entries during program execution. It is initialised with the ON KEY (n) statement.

KEY (n) ON, when executed, activates trapping of the keys. A GOSUB is automatically executed when a key is pressed to the line number specified in the ON KEY (n) statement.

The display of the soft key function is not altered with this statement.

If the same key is pressed again while the subroutine is being processed, then the interrupt is stored until the program returns from the interrupt routine when the GOSUB will be executed again.

If a different key is pressed while one interrupt routine is being processed, then another GOSUB is immediately executed. Different key trapping routines may be nested, but each individual routine cannot be re-entered.

KEY (n) OFF The trapping routine is deactivated, and the key press is ignored.

KEY (n) STOP Immediate execution of the trapping routine does not take place, but the key is remembered and left pending until a KEY (n) ON statement is encountered.

KILL

5.12.3 **Function** KILL deletes a file in RAM storage.

Syntax KILL filename

Example KILL "FIRST.HBA"

Deletes FIRST.HBA

KILL A$

Deletes the filename contained in A$

Remarks In order to KILL a file the filename must be given either explicitly or only using the "?" wildcard. Variable names are accepted, but must contain actual filenames.

NOTE: If the file does not exist, then no action is taken and no warning message issued.

If wildcards are used, then all files which conform to the format are erased, e.g. if H.1 and H.2 exist as files, then:

KILL "H.?"

will delete both of them.

LEFT$

5.13.1 **Function** **LEFT$** returns the first n characters from a designated string. Either string or n may be expressions.

Syntax LEFT$("STRING",N)

where string is a string expression N is the number of left most characters.

Examples
```
10 A$ ="ABCDEFG"
20 B$ = LEFT$(A$,3)
30 PRINT B$
```

prints ABC.

Remarks LEFT$(A$,0) returns a Null String i.e. ""
LEFT$(A$,50) returns ABCDEFG

N cannot be negative or a "MAG" error will result.

LEN

5.13.2 **Function** **LEN** returns the length of a string.

Syntax LEN (string)

where string is a string expression.

Examples
```
10 A$ = "ABCDEFG"
20 PRINT LEN (A$)
```

prints 7.

Remarks The string may be an expression.

A null string has zero length.

LET

5.13.3 **Function** **LET** assigns a value to a variable.

Syntax LET variable name = numerical expression
LET string variable = string expression

Examples
```
LET C = 5
LET D3 = B
LET J = SIN(X)
LET A$ = "HELLO"
LET A$ = B$ (see note)
```

Use of LET in an assignment statement is optional. The formats:

```
C = 5
D3 = B
J = SIN(X)
A$ = "HELLO"
A$ = B$  (see notes below)
```

are equally acceptable.

Remarks **NOTE:** When equating string arrays of unequal length the string array on the right side of the equates will be truncated on the left side.
Example:

```
DIM A$(1,5)
A$ = "HELLO"
B$ = "GOODBYE"
A$ = B$
```

then A$ = "GOODB"

NOTE: Remember that string lengths default to 20 characters. In the above example, A$ should be DIMed to 5 character length as shown.

Strings can be created by multiple additions.
Example:

```
A$ = A$ + MID$(C$,4,3)+CHR$(65)
```

String expressions can also be created from complex string expressions of up to 10 functions. The derivation of the function requires the creation of an expression stack which can handle a maximum of ten functions. If the stack is exceeded a 'string complexity' error will result.

Example:

```
A$ = LEFT$(RIGHT$(MID$(A$,3,N),A),B)
```


LINCHR

5.13.4 **Function** **LINCHR** inputs a single character from the RS-232 port.

Syntax
```
LINCHR variable name
LINCHR ["PROMPT STRING";] VARIABLE NAME
```

Examples
```
10  LINCHR "DECIMAL VALUE",A
20  PRINT A
30  GOTO 10
```

displays:

```
DECIMAL VALUE  65
DECIMAL VALUE  66
etc.
```

Remarks LINCHR does not wait for carriage return to be received and the character input is not echoed to the screen. A prompt message is displayed on the LCD screen as with INPUT. Only one variable will be received. The variable is set to the decimal value of the character, as detailed in section 9.2, ASCII CHARACTER SET.

A comma following the prompt message will supress the prompt character '?'.

Characters input from the serial port are not echoed onto the screen.

LINE

5.13.5 **Function** **LINE** draws lines and boxes on the graphics screen.

Syntax Line [(X1,Y1)] - (X2,Y2)[,P[,B[F]]] 0<X<239 0<Y<63

Examples 10 LINE (10,10) - (50,60)

Draws a line from the co-ordinates (10,10) to (50,60)

Remarks The parameter P is optional and if included denotes the colour of the line. If p is odd then the lines are drawn in black and if it is even then it is drawn in white (i.e. erased). P defaults to odd if omitted.

The parameter B is used for drawing boxes with (X1,Y1) and (X2,Y2) specifying the co-ordinates of the two diagonal corners.

The F parameter is used to fill in the box with the specified colour P. The P parameter must be specified if either the B or BF is used. The following command will fill the graphics screen with a black filled in box:

```
LINE(0,0)-(239,63),1,BF
```


LINPUT

5.13.6 **Function** **LINPUT** obtains a line of data from HUNTER's RS-232 port.

Syntax LINPUT [;]["prompt string";] variable name

Examples LINPUT A

Inputs from the serial port a numeric value and places it in variable A.

LINPUT "PARAMETER VALUES" A,B,C

Input the values of A, B and C with the prompt message:

```
PARAMETER VALUES
?
```

Remarks Note that input lines are always terminated on reception of a carriage return (CR) character.

LINPUT cannot be used as a direct statement.

A prompt message can be displayed by the LINPUT statement. A string variable name or a literal string is required as an argument. Multiple variables can be strung together, separated by commas.

The characters input from the serial input port are NOT echoed to the screen.

A comma following the prompt message will suppress LINPUT's '?' prompt.

NOTE: If LINPUT's argument is a simple variable and an invalid entry is made, LINPUT will simply re-prompt without an error message. An example is:

```
10 LINPUT A
```

HUNTER receives 'HELLO'. After 'CR', LINPUT generates a loud 'bleep' and re-prompts on the next line.

NOTE: While the HUNTER is waiting for the first character to be received from the port, it will 'bleep' at 30 second intervals.

LIST

5.13.7 **Function** LIST causes HUNTER to list the program on the LCD screen.

Syntax LIST [line no1][-line no2]

Examples LIST 100-

Lists from line 100 on the screen. If there is no line 100 it will LIST from the next line in sequence.

Remark LIST is optionally followed by a start line number and a finish line number, either of which may be omitted. If both are omitted the entire program will be listed.

LLIST

5.13.8 **Function** **LLIST** causes HUNTER to list the Basic program to the serial port.

Syntax LLIST [line no.1][-line no.2]

Examples LLIST 100-

will cause a listing to start at line 100 in the program.

Remarks LLIST is optionally followed by a start line number and a finish line number, either of which may be omitted. If both are omitted, the entire program will be listed.

LLIST can be terminated at any time by the escape ('ESC') key being depressed and held down until HUNTER responds with a confirmatory 'beep'. Because a transmission buffer is used, up to 256 characters may remain for transmission after 'ESC' is confirmed. Power OFF will also abort a listing at any time without loss of stored data or program.

It is advisable to check HUNTER's Communication Parameters, see section 6.4, COMMUNICATION PORT SOFTWARE, before using LLIST for compatibility with the external device receiving the data. When a printer is being used, remember to check for baud rate and line feed requirements. Unless "Echo" is enabled in transmission parameter, there is no data echoed onto the screen.

NOTE: Because the communications power supply is powered up by LLIST, a single high-to-low character will appear at the start of the record. Many systems will see this transition as a single, spurious, character. To avoid this problem, power up the interface first with either:

LPRINT "" or OUT 132,1

otherwise, it may be necessary to edit out this character on the host system.

LLOAD

5.13.9 **Function** **LLOAD** command allows the HUNTER to load Basic programs through its serial port.

Syntax LLOAD

Examples LLOAD

Loads Basic source from the RS-232 port.

Remarks Programs which were output using LLIST onto a storage medium, e.g. disk or another HUNTER may be reloaded using this command. LLOAD is terminated by typing ESC. See section 3.6, LOADING FILES, for a detailed description of program loading/unloading options.

Be sure to check HUNTER's receiving parameters for compatibility with the device transmitting data.

LLOAD data goes into memory via the Basic interpreter input routine and is syntax checked on the way - if errors or spurious data are found in the incoming data, HUNTER will stop loading and display:

* SYX Error

If this occurs at the start of a file, it is likely that some header data, not in Basic syntax, has caused the problem.

Simply type LLOAD again, quickly, as HUNTER's input buffer will still be receiving data.

LLOAD does not echo incoming characters onto the screen.

LN

5.13.10 **Function** **LN** generates the NATURAL logarithm of X, i.e. logarithm to base e of X.

Syntax LN(N)

where N is a numeric expression.

Examples LN(8)

generates the logarithm to base e of 8. (2.079441601991)

If S=9.12 then:

LN(S)

generates the logarithm to base e of 9.12.(2.2104698042074)

LN(-91)

generates a 'Magnitude Error' and program execution will stop.

Remarks X must be greater than zero.

If X is negative or equal to zero then a 'Magnitude Error' will occur.

X may be an expression.

LOAD

5.13.11 **Function** **LOAD** reads a program into the Basic workspace from a file.

Syntax LOAD filename [,R]

Example LOAD "HUNTER.HBA"

loads the Basic file, HUNTER.HBA into the workspace ready for execution.

```
LOAD "HUNTER",R
```

loads the Basic file HUNTER.HBA into the workspace and commences execution.

```
LOAD "HUNTER.TXT"
```

This loads an ASCII file into the Basic workspace. It processes the ASCII into Basic compressed format.

Remarks For .HBA files, typing in the extension is optional.

LOC

5.13.12 **Function** **LOC** returns the number of records read/written to a sequential file

Syntax LOC(file no)

Example

```
100 MAXFILES=2
110 OPEN "TEST"FOR INPUT AS #2
120 INPUT#2,A
130 IF EOF(2)=0 THEN GOTO 120
140 PRINT"End of File"
150 PRINT"No. of records read = ";LOC(2)
160 CLOSE#2
170 END
```

When the end of file has been reached the number of records that have been read will be displayed.

Remarks The LOC(n) function can also be used to display the number of records written to a file.

NOTE: the program to create the data file TEST could be of the form:

```
10 OPEN "TEST" FOR OUTPUT AS#1
20 FOR X=1 TO 100
30 WRITE#1,X
40 NEXT
50 CLOSE#1
60 END
```

NOTE: A record consists of 128 bytes, so LOC gives a measure of the amount of file space used.

LOCATE

5.13.13 **Function** **LOCATE** sets the text cursor on the screen.

Syntax LOCATE x,y

Maximum Value

Character Set	x	y
Text mode	79	23
Char 0	239	63
Char 1	30	5
Char 2	30	5
Char 3	30	5
Char 4	30	5

Example 10 LOCATE 0,0

Places the cursor in the top left-hand corner.

Remarks LOCATE is most useful for locating the cursor before printing text onto the display. LOCATE operates differently in text mode and graphics mode.

Text Mode: Addressing a location outside the current window will cause the window to move so that the new cursor position is visible.

The maximum values for the parameters are X = 79 and Y = 23, being the size of the virtual screen.

Values greater than the maximum for X will cause wrap-around on the same line. Values greater than the maximum for Y will remain at the bottom of the screen.

Graphics Mode: A SCREEN 1 command will reset the cursor to the origin at (0,0). When character set zero is selected the LOCATE co-ordinates are referenced to the graphics pixel position. Otherwise, the co-ordinates specified in LOCATE refer to the top left hand pixel of a 7 x 5 character cell, the size of which is equivalent to a CHAR1 character.

LOG

5.13.14 **Function** **LOG** generates the logarithm to base 10 of X.

Syntax LOG(N)

where N is a numeric expression.

Examples LOG(32)

generates the logarithm to base 10 of 32.

If S=1.319 then:

LOG(S)

generates the logarithm to base 10 of 1.319.

LOG(-6)

generates a 'Magnitude Error' and programme execution will stop.

Remarks X must be greater than zero.

If X is negative or equal to zero then a 'Magnitude Error' will occur.

X may be an expression.

LOPCHR

5.13.15 **Function** **LOPCHR** outputs characters to the RS-232 port which have their ASCII values defined as decimal argument(s).

Syntax LOPCHR numerical exprssion 1, numerical expression 2...

Examples LOPCHR 61,62 displays =>

Remarks This permits computing of characters and also advanced functions such as cursor addressing. The argument should be 0 - 225.

LPRINT

5.13.16 **Function** **LPRINT** outputs to the serial port.

Syntax LPRINT expressions.

Examples LPRINT A,B,C$

Outputs variables A,B and C$ to the serial output port.

Remarks LPRINT is exactly equivalent to PRINT except that output goes to the serial RS-232 port and not the screen.

See PRINT for a full description.

LPRINT formats can be altered by the following statements:

LPRINT % N %

sets the number of decimal places equal to N, a number in the range 0-9. Rounding occurs up or down as appropriate.

LPRINT % Z N %

sets the number of trailing zeroes (or decimal places) equal to N, a number in the range 0-9.

LPRINT % E %

sets all following LPRINT formats to scientific notation.

LPRINT % %

restores LPRINT format to normal

LTRON

5.13.17 **Function** **LTRON** turns on the trace but sends the trace output to the printer.

Syntax LTRON

Examples LTRON

Remarks LTRON, terminated with TROFF, produces the same output as TRON but on an external printer.

See TRON for a full description.

MAXFILES

5.14.1 **Function** **MAXFILES** defines the maximum number of files that may be opened simultaneously.

Syntax MAXFILES = n

Examples MAXFILES = 4

Sets the maximum number of files that may be opened simultaneously to four.

Remars MAXFILES will perform a 'CLEAR' operation destroying all existing symbol table entries. It must therefore be defined at an early point in the program.

MAXFILES may also be used for input, for example:

```
100 PRINT"Maximum number of files to be opened = ";MAXFILES
```

will display:

```
Maximum number of files to be opened = 4
```

If no MAXFILES has been specified a default value of one is taken.

The maximum number of files that may be opened simultaneously is 15.

MID$

5.14.2 **Function** **MID$** returns a substring of given length starting at the given position. It may also be used as a statement.

Syntax B$=MID$ (A$,X,[Y])

or

MID$ (A$,X,[Y]) = C$

Examples
```
10 A$= "ABCDEFGH"
20 PRINT MID$ (A$,3,4)
Prints CDEF
```

Remarks A$,X or Y may be expressions. If Y is omitted then all characters to the right of the Xth character are returned. In the above example:

MID$ (A$,0,3) = 'ABC'
MID$ (A$,20,5) = null string
MID$ (A$,7) = 'GH'

When MID$ is used as a statement in the form:

MID$(A$,X[,Y])=C$

its sets the part of A$ starting at the Xth character with the substring C$. The optional parameter Y denotes the length of the part of A$ to be replaced and if omitted defaults to the length of C$.

NAME

5.15.1 **Function** **NAME** renames a HUNTER file.

Syntax NAME oldname AS newname

Example NAME "VERSION1.HBA" AS "VERSION2.HBA"

NAME A$ AS B$

Renames the file VERSION1.HBA to VERSION2.HBA

Remarks After the NAME command, only the filename is altered. Filenames can be up to eight characters long with a three character extension.

NEW

5.15.2 **Function** NEW will re-initialise all HUNTER's user memory and reset all program pointers.

Syntax NEW

Examples
```
READY
NEW

Are you sure (Y/N)? Y

54613 bytes available

READY
```

Remarks All program lines are erased!

NEW should, therefore, not be used if several programs are to co-exist in the same HUNTER.

Always use NEW before loading a new user program into HUNTER to ensure the maximum use of space.

NEW replies with:

    Are you sure? (Y/N)

If any response to the confirmation message other than YES (Y) is made, then the function is aborted and existing programs are not affected.

Following the 'Y' response, NEW displays the number of free bytes available before returning to Basic.

Filespace is not affected by 'NEW'.

NEXT

5.15.3 **Function** **NEXT** is the last statement in a FOR.....NEXT loop.

Syntax NEXT [V]

V is optional and is a variable name that corresponds to the original FOR statement.

Examples NEXT I or NEXT

ON BREAK

5.16.1 **Function** **ON BREAK** allows the user to interrupt processing by operation of the BRK (Break) key.

Syntax ON BREAK GOTO [line]

or

ON BREAK OFF

Examples
```
ON BREAK GOTO 1000
ON BREAK OFF
```

Remarks This statement operates in a similar manner to the ON POWER statement, except that the break key is monitored instead of the power key.

ON COM

5.16.2 **Function** **ON COM** defines a routine to be executed when data is received into the communications buffer.

Syntax ON COM GOSUB line number

Example
```
10 ON COM GOSUB 100
20 COM ON


100 'SUBROUTINE TO READ
110 'INCOMING DATA


190 RETURN
```

Remarks This statement is used in conjunction with the COM statement. It tells Basic where to jump to in the event of a communications interrupt.

Data may be left in the communications buffer for as long as is necessary.

On an interrupt an automatic COM STOP is executed so interrupts are never nested. The RETURN from the trap routine performs an automatic COM ON, so re-activating the interrupt routine, unless a COM OFF statement was executed before the RETURN.

ON COMMS

5.16.3 **Function** **ON COMMS** allows communication errors (typically a protocol failure) to be handled by the user's application program.

Syntax ON COMMS GOTO [line no]

or

ON COMMS OFF

Examples
```
ON COMMS GOTO 1000
ON COMMS OFF
```

Remarks The statement operates in a similar manner to ON POWER.

When a communications error is detected, control is returned to the Basic interpreter. Execution continues from a defined line number specified in the Basic program.

In order to allow normal communication error messages the statement ON COMMS OFF disables this mode. The byte of memory COMERR is available to the programmer and contains a code which represents the type of error which has occurred in the communications. The available codes and their meaning are described more fully in section 6.7.8, COMMUNICATION ERROR.

In a typical application, ON COMMS would be used to present the Operator with a plain language statement of the form:

"Communications have failed. Please re-dial and try again".

Re-trys or repeat blocks can also be controlled by the user program, as required.

ON ERROR

5.16.4 **Function** **ON ERROR** is used to catch errors which would normally give a Basic Error message.

Syntax ON ERROR GOTO [line no]

ON ERROR GOTO 0

Examples
```
10 ON ERROR GOTO 1000
20 INPUT A
30 A = SQR(A)
40 PRINT A
50 GOTO 20
1000 A =ABS(A)
1010 PRINT "NEGATIVE NUMBER CORRECTED"
1020 RESUME 30
```

This example will trap negative numbers and correct the condition.

Remarks ON ERROR causes program execution to be diverted to an error handling subroutine.

Any error can be trapped. However, ON ERROR is normally used for correctable errors e.g. magnitude errors which could be caused by operator input etc.

An ON ERROR trap must be exited by either an RESUME statement or by ON ERROR GOTO 0 which will give the normal Basic error message.

ON..GOSUB

5.16.5 **Function** **ON...GOSUB...** allows conditional subroutine calls.

Syntax ON expression GOSUB line number 1, line number 2, ...line number n.

Examples

```
10 ON A GOSUB100,180
20 PRINT "HELLO"
```

If A=1 then program execution will go to line 100 and will then go to line 20 when a RETURN is encountered.

If A=2 then program execution will go to line 180 and will then go to line 20 when a RETURN is encountered.

```
10 B=100:C=300:D=350
20 ON A GOSUB B,C,D
30 PRINT "HELLO"
```

If A=1 then program execution will go to line 100 and revert back to line 30 when a RETURN is encountered.

If A=3 then program execution will go to line 350 and revert back to line 30 when a RETURN is encountered.

Remarks ON GOSUB will cause program execution to go to a line number chosen from the list of line numbers positioned immediately after GOSUB. The selection of a particular line number in this list is determined by the value of [expression].

Once program execution has been forced to go to one of the selected line numbers, each statement following that line number will be executed in order until a RETURN is encountered. At this point program execution will revert back to the statement following the list of line numbers.

The integer part of [expression] is used so that if A=2.9 program execution will go to line 180 in the example.

NOTE: The value of [expression] is **NOT** rounded up.

If the value of [expression] is zero or greater than the number of line numbers in the list, then program execution will continue with the next statement.

In the above example, if A=3 then "HELLO" will be printed **IMMEDIATELY.**

The maximum value of [expression] allowed is 255.

If the value of [expression] is greater than 255 or negative then a 'Magnitude error' will occur.

The line numbers in the list of line numbers may themselves be expressions.

ON..GOTO

5.16.6 **Function** **ON...GOTO...** allows conditional jumping.

Syntax ON expression GOTO line number 1, line number 2, ...line number n.

Examples
```
10 ON A GOTO100,200,300
20 END
```

If A=1 then the program will go to line 100
If A=2 then the program will go to line 200
If A=3 then the program will go to line 300

If A=1.3 then the program will go to line 100.

Remarks ON GOTO will cause program execution to jump to a line number chosen from the list of line numbers positioned immediately after GOTO. The selection of a particular line number in this list is determined by the value of [expression].

The line numbers may themselves be expressions.

The value of [expression], in the above example A, is truncated to an integer.

It must be noted that the value is truncated to an integer and **NOT** rounded up. So if the above example A=1.9 then the program will go to line 100.

If the value of [expression] is zero or greater than the number of line numbers in the list, then program execution will continue with the next statement.

In the above example, if A=4, or if A=0 then the program will automatically go to line 20, and an END will be executed.

The maximum value of [expression] allowed is 255.

If the value of [expression] is greater than 255 then a

'Magnitude Error' will occur and program execution will stop. If in the above example A=256 then an error will occur. A 'Magnitude Error' will also occur if the value of [expression] is negative.
Example:

```
10 ONAGOTO 100,200,300:PRINT "HELLO"
20 END
```

If A>3 or A=0 then the program will not go to any of the lines but will immediately print "Hello", and then END will be executed.

ON KEY

5.16.7 **Function** **ON KEY** specifies a routine to be executed when a soft key or cursor control key is pressed during program execution.

Syntax ON KEY (n) GOSUB line number

Examples ON KEY(1) GOSUB 5000

If function key 1 is pressed the program will execute the subroutine located at line 5000.

Remarks For a definition of n and details of using this function see the KEY (N) statement. Line number specifies the trap routine, which must be terminated with a RETURN.

ON POWER

5.16.8 **Function** **ON POWER** allows the programmer to trap and vector program execution if a user presses the power off key.

Syntax ON POWER GOTO [line no]

or

ON POWER OFF

Examples ON POWER GOTO 1000

Remarks This statement causes the OFFVECT locations to be loaded with a vector so that if operation of the power down key is detected, program execution occurs from the line number specified.

Operation of the break key, or a syntax error, will cause the vector to be re-initialised to enable the power on/off switch to function normally.

ON POWER OFF

This statement also re-initialises the OFFVECT location.

ON POWER RESUME

5.16.9 **Function** **ON POWER RESUME** restarts program

Syntax ON POWER RESUME

Examples ON POWER RESUME

Remarks After HUNTER has been switched off, either by the power off key or any other means, switching HUNTER on will restart the program from the point where it was switched off, even if input was in progress.

Exit from ON POWER RESUME is made by holding down 'control and C' before pressing the power on key and entering the escape code, see section 3.8.4, Exiting 'CONT', for details of entering the escape code.

ON TIME$

5.16.10 **Function** **ON TIME$** specifies a routine to be executed at a defined time.

Syntax ON TIME$ = A$ GOSUB line number

A$ is a string defining the interrupt time

Example 10 ON TIME$ = "12:00:00" GOSUB 1000

Remarks The executing program will be interrupted at the specified time and execution will begin again in the subroutine beginning at 'line number'.

The time is checked at the beginning of a new statement, therefore if Basic is waiting on an input, for example, the interrupt will not occur immediately.

NOTE: ON TIME$ is only effective when HUNTER is powered up.

OPCHR

5.16.11 **Function** **OPCHR** outputs characters to the screen which have their ASCII values defined as decimal argument(s).

Syntax OPCHR numerical exprssion 1, numerical expression 2...

Examples OPCHR 61,62 displays =>
OPCHR1 clears the screen.

Remarks This permits computing of characters and also advanced functions such as cursor addressing. The argument should be 0 - 255. Arguments greater than 255 will give an argument error.

Cursor addressing:

Example: OPCHR 15,5,3

The argument 15 instructs HUNTER Basic to interpret the following two arguments as cursor co-ordinates x,y respectively. It must be noted that re-positioning the cursor in this manner will not cause the virtual screen window to be re-positioned over the relevant area of the virtual screen automatically. If automatic re-positioning is required then use LOCATE.

OPEN

5.16.12 **Function** **OPEN** opens file for I/O

Syntax OPEN "file name" FOR OUTPUT / APPEND / INPUT AS file no.

Example 10 OPEN "OUTFIL.TXT" FOR OUTPUT AS 1

This will open the file "OUTFIL.TXT" for output from a Basic program as file number one.

or 10 OPEN "APPEND.TXT" FOR APPEND AS 3

This opens the file "APPEND.TXT" as file number three. Any data written to the file will be appended. MAXFILES must have previously been set to a minimum of three.

or 10 OPEN "INPUT.TXT" FOR INPUT AS 4

The file is opened for input as file number four, this will allow the data in the file to be read by a Basic program.

Remarks When a file is OPENed HUNTER's firmware automatically performs a checksum generation on the file and compares it to a value stored in the file directory, when the file was last CLOSEd. Any inconsistency will result in a 'DSK ERROR' being returned to the user.

The OPEN statement parameter FOR has three options OUTPUT,APPEND and INPUT, they perform the following functions:

OUTPUT : This will open a file for output. Any data that is in the file will be lost.

APPEND : This will also open a file for output, but any data that is output to the file will be added to the end of the data that is already present.

INPUT : The file is opened for input. This will allow a Basic program to read data from the file.

The file number cannot be greater than the number specified in MAXFILES. If MAXFILES 3 has been specified the file number cannot be greater than three.

OUT

5.16.13 **Function** **OUT** outputs a byte of data to a specified port.

Syntax OUT address, data 1, data 2....data n.

Examples OUT 133,1

Switches DTR to the off state.

Remarks OUT is used to control internal HUNTER functions like power control. See section 9.8, PORT ALLOCATIONS, for details of port addresses.

PEEK

5.17.1 **Function** **PEEK** fetches the value of a designated byte in HUNTER's memory.

Syntax PEEK (ADDRESS)

where ADDRESS is a numeric expression.

Examples PRINT PEEK (17933)

prints the value of location 17933 (in decimal) on the LCD screen.

Remarks PEEK requires the decimal value of the required memory location as an argument. See also DEFSEG to set the page address.

PI

5.17.2 **Function** PI is used to return the value of PI

Syntax A = PI

Example
```
10 INPUT "ANGLEIN DEGREES",A
20 LET B = A*PI/180
30 PRINT A;"DEGREES ARE";B;"RADIANS"
```

This program will convert an angle in degrees to an angle in radians.

Remarks PI is returned with 6 digit precision i.e.

PI = 3.14159

POINT

5.17.3 **Function** POINT returns the 'colour' of the specified pixel.

Syntax A = POINT (X,Y)

Example
```
10 REM Invert the graphics screen
20 FOR X = 0 TO 239
30 FOR Y = 0 TO 63
40 IF POINT(X,Y) = 1 THEN PRESET (X,Y) ELSE PSET (X,Y)
50 NEXT Y:NEXT X
```

Remarks An odd number returned by the function indicates a black pixel, while an even number indicates a white pixel. If the co-ordinates are out of range then -1 is returned.

POKE

5.17.4 **Function** **POKE** allows the user to write to a specified location in HUNTER's memory.

Syntax POKE address, data 1, data 2,....data n.

Examples POKE AD,1

Pokes the value 1 into location AD.

Variable AD contains the address to be POKEd.

Remarks The data can be a variable or an expression and must be in the range 0-255. Values greater than 255 will give an argument error. See also DEFSEG to set the page address.

WARNING Use POKE with care.

POP

5.17.5 **Function** **POP** fetches (POP's) a value from the machine code subroutine linkage stack.

Syntax POP(N)

where N=number of parameters to be POPPED.

Examples A=POP(2)

POPs two values off the machine code linkage stack.

A=latter parameter.

Remarks Allows Basic to access 16 bit results from machine code subroutines.

Refer to section 4.8, MACHINE CODE CALLS, which discusses machine code calls.

WARNING do not attempt to pop more variables from the stack than have been pushed.

POS

5.17.6 **Function** POS returns the current cursor position on the line.

Syntax POS(n) n is a dummy argument.

Examples A=POS(X)

Variable 'A' contains the current cursor column position.

Remarks The returned value will be in the range 1 to 80.

POWERn

5.17.7 **Function** POWER n auto power off period.

Syntax POWER n

n is in the range 10-255 and is the number of 5 second periods.

Examples POWER 12

will switch HUNTER off after 1 min.

POWER 0

will prevent timed switch off.

Remarks POWER 0 will enable the power off key after POWER CONT has been executed.

To remind the user that HUNTER has been left switched on. HUNTER will emit a bleep every 150 seconds. Power n is reset to the default value (n=60, or 5 minutes) after power off/on and should be re-initialised by the user's program.

POWER CONT

5.17.8 **Function** **POWER CONT** disables the power off key and time outs.

Syntax POWER CONT

Examples POWER CONT

This will disable the power off key and time outs.

Remarks To enable the power off key and time outs use POWER n.

Use of POWER CONT allows the user program control of the power key to force, for instance, completion of a data entry sequence before the unit can be powered down.

POWER OFF

5.17.9 **Function** **POWER OFF** switches HUNTER off

Syntax POWER OFF

Examples POWER OFF

Remarks After POWER OFF has been executed switching HUNTER on will automatically run the program from the start.

To exit from POWER OFF press 'Esc' and enter the code 56580.

Power off is used in applications where 'Cont filename' could give confusing results if the unit is left in an uncontrolled state by a previous user.

POWER OFF RESUME

5.17.10 **Function** **POWER OFF,RESUME** switches HUNTER off.

Syntax POWER OFF,RESUME

Examples POWER OFF,RESUME

Remarks After POWER OFF,RESUME has been executed switching HUNTER on will re-start the program from the next statement and restore the screen.

Unlike POWER OFF, this function does not require the code 56580 after 'Esc' in order to exit from the program.

PRINT

5.17.11 **Function** **PRINT** displays data on the LCD screen.

Syntax PRINT expressions [;]

or ? expressions

The question mark is exactly equivalent to PRINT

Example PRINT X,Y,Z$

Prints the values of the three variables X,Y, and Z$ in columns 1, 14, and 28 and finishes with CR/LF. A PRINT statement with no expressions just performs the CR/LF functions.

Remarks The comma between the expressions move the print cursor to the next zone, where each zone is fourteen spaces wide. The use of semi-colons causes the values to be printed together without gaps. However, numerical data leaves a space at the start for the minus sign, and includes one blank space afterwards. A semi-colon at the end of the line suppresses the CR/LF.

Examples:

```
PRINT "FOOT" + "BALL"          outputs FOOTBALL
```

or

```
PRINT X;"SQUARED IS";          outputs, if X=2
PRINT X*X                      2 SQUARED IS 4
```

or

```
IF A<2 THEN PRINT "A=";A       outputs, if A=1
                               A=1
```

or

```
PRINT CHR$(1);                 will clear the LCD display.
```

This page intentionally left blank.

This page intentionally left blank

PRINT#

5.17.13 **Function** **PRINT#** writes data to file.

Syntax PRINT#file no,var1;var2;....varX

Examples PRINT#4,A$;B4;D$

or

PRINT#1,X;Y;Z

The variables specified are written to the appropiate file number.

Remarks Variables should be separated by semicolons and not commas. As PRINT# outputs to the file in the same way as it would to the screen, any spaces present if commas were used would also be written to the file.

Care should be taken when writing string data to files as they should not contain any commas, semicolons, leading blanks, carriage returns or line feeds as they would be interpreted as delimiting characters. If it is necessary to include any delimiting characters in a string the variable name should be enclosed in double quotes by using CHR$(34).

For example, if A$ ="TOM,DICK" and B$="HARRY". The statement:

PRINT#2,A$;B$

would write the following to the file:

TOM,DICKHARRY

and the statement:

INPUT#2,A$,B$

would input 'TOM' to A$ and 'DICKHARRY' to B

To separate the data correctly the following format should be used:

PRINT#2,CHR$(34);A$;CHR$(34);CHR$(34);B$;CHR$(34)

This would write the following to the file:

"TOM,DICK""HARRY"

and the statement:

INPUT#2,A$,B$

inputs 'TOM,DICK' to A$ and 'HARRY' to B$

The statement WRITE#n is a simpler way to perform the above functions.

PSET/PRESET

5.17.14 **Function** **PSET** and **PRESET** are used to set or reset an individual pixel at the specified coordinates.

Syntax PSET (X1,Y1),(X2,Y2),(X3,Y3),...

PRESET (X1,Y1), (X2,Y2),...

Any number of points may be specified in one statement.

(x,y) are the coordinates of the pixel to be set or reset and must lie in the range 0<x<239 and 0<y<63. The coordinate (0,0) refers to the top left corner of the screen.

Examples PSET(0,0),(239,0),(0,63),(239,63)

Will set the four corner pixels black.

Remarks PSET and PRESET are graphic statements and will default the screen to graphic mode. If when a PSET/PRESET statement is encountered the screen is in text mode, it will automatically change to graphic mode.

PSET sets pixels (turns them black)
PRESET resets pixels (turns them white)

Each time a pixel is set or reset using PSET and PRESET the graphic cursor is moved to that pixel.

PUSH

5.17.15 **Function** **PUSH** pushes a 16 bit value onto HUNTER's machine code subroutine linkage stack which is generally used with machine code **CALL's**.

Syntax PUSH variable 1, variable 2, ...variable n.

Examples PUSH A,B

pushes the integer (16 bit binary) representation of the variables A and B onto the stack.

Remarks These variables may be accessed or modified by a machine code program. See section 4.8, MACHINE CODE CALLS.

POP is the reverse of PUSH.

READ

5.19.1 **Function** **READ** is used to input constants previously defined in a DATA statement.

Syntax READ Variable 1, Variable 2,...Variable n

Examples READ X

sets X equal to the current DATA constant.

For example the program:

```
10 DIM D!(10):DIM A$(10,10)
20 DATA 3,4,5,2/3,"HELLO"
30 READ A,B,C,D!(0),A$(1)
40 PRINT A;B;C;D!(0);A$(1)
```

Prints:

```
3 4 5  .66666666666667 HELLO
```

with A,B,C equal to three sequential DATA constants, D!(0) a double precision constant and A$(1) a string expression.

Remarks The constants are read sequentially until exhausted, when an error will result. The sequence may be restarted by using RESTORE.

The variable type in READ **must** be the same as the corresponding DATA items: if not, a READ ERROR will occur.

REM

5.19.2 **Function** **REM** denotes that the following text is a comment statement only.

Syntax REM text

or

' text

Example

```
1000  REM THIS ROUTINE CLEARS THE DISPLAY
1010  REM
1020  PRINT CHR$(1);
1030  RETURN
```

Remarks REM statements are ignored by the program. They can considerably improve readability but only at the expense of extra memory. Unnumbered REM statements can be used in host computer files which will be ignored when down loaded.

The apostrophe may be used instead of 'REM', but has an extra capability. When delimiting a comment after a Basic statement, the colon is unnecessary.

```
10  X=X+1' THIS TYPE OF COMMENT SAVES MEMORY SPACE
```

REM statements are also used as HELP source text.

RESTORE

5.19.3 **Function** **RESTORE** is used with READ. RESTORE initialises the current DATA pointer to the first DATA statement in the program.

Syntax RESTORE [line number]

Example RESTORE 120

This restores the data pointer to line 120.

Remarks RESTORE may also be used with a line number argument. This will initialise further READ functions to a specified line.

RESUME

5.19.4 **Function** **RESUME** restarts program execution at the specified point after an error.

Syntax RESUME [NEXT] [line number]

Example

```
10 ON ERROR GOTO 100
20 IF A<B THEN ERROR 110
"
"
"
100 IF ERR>100 THEN RESUME L2+(ERR/10)
```

Remarks If there is no argument in the statement then execution will begin again at the statement which caused the error.

RESUME NEXT begins on the following statement and RESUME [line number] begins at the specified line.

RETURN

5.19.5 **Function** **RETURN** from subroutine.

Syntax RETURN

Examples 10 GOSUB 100

```
100 PRINT I
200 RETURN
```

Remarks RETURN is the logical conclusion of a GOSUB statement. Placed at the end of the subroutine, RETURN causes execution to continue at the next statement following the original GOSUB statement.

RIGHT$

5.19.6 **Function** **RIGHT$** returns the last n characters of a given string.

Syntax RIGHT$("STRING",N)

where N is the number of rightmost characters to be returned.

Examples RIGHT$("ABCDEF",3)

returns DEF

Remarks Both string and n must be enclosed in parentheses and can be expressions. If length of the string is less than n then the entire string is returned.

If A$ = "ABCDEF"

then:

RIGHT$(A$,0) = null string
RIGHT$(A$,20) = ABCDEF

RND

5.19.7 **Function** **RND** returns a 14-digit psuedo random number in the range 0 - 0.99999999999999.

Syntax RND(0)

where "0" is a dummy argument.

Examples A = RND(0)

sets A equal to a random number.

Remarks RND requires a dummy argument.

RUN

5.19.8 **Function** **RUN** causes program execution to commence.

Syntax RUN [line number] [numerical expression] [filename]

Examples RUN 1000

causes execution to commence from line 1000.

Remarks RUN is useful when program de-bugging as it initialises the system stacks, GOTO does not.

The line number may also be a variable or an expression, for example:

RUN A
RUN A*100

or

RUN "filename"

causes the "filename" to be loaded and executed from the work area.

RUN is a direct Basic command only.

SAVE

5.20.1 **Function** **SAVE** writes a program into the file space.

Syntax SAVE filename

Example SAVE "HUNTER.HBA"

Remarks The Basic program is written to the filespace in RAM. If a file already exists with the same filename, then the old file is overwritten. Filenames are up to eight characters long. The extension .HBA is automatically appended for a Basic program, if an alternative is not provided.

An appropriate error message is displayed if there is insufficient file space for the program file.

SCREEN

5.20.2 **Function** **SCREEN** changes the mode of operation of the LCD screen.

Syntax SCREEN n

n may be any numeric expression returning either a 0 or 1. Any other value will give a Magnitude Error.

Example SCREEN A-A will put the screen into text mode.

SCREEN 1 will put the screen in graphic mode.

Remark TEXT MODE (SCREEN 0)

This is the main mode of operation for the HUNTER. This mode is used to list Basic programs and to enter programs. In this mode the output is purely textual and gives a window of 8 lines of 40 characters on a virtual screen of 24 lines of 80 characters.

GRAPHIC MODE (SCREEN 1)

In this mode the user is given much more control over the screen with high resolution graphics and up to 5 different types of character set.

NOTE: Graphic commands set SCREEN 1 mode automatically.

SGN

5.20.3 **Function** SGN returns the value of the sign of the argument.

Syntax SGN(N)

where N is a numeric expression.

Examples A = SGN(-9.5)

sets A equal to -1

A = SGN(0)

sets A equal to 0

A = SGN(1.76E9)

sets A equal to 1

Remarks Result:

1 if Positive
0 if Zero
-1 if Negative

SIN

5.20.4 **Function** SIN returns Sine of the argument.

Syntax SIN(N)

where N is a numeric angle in radians.

Examples J = SIN(2.56)

sets J equal to the value of the sine of 2.56 radians.

PRINT SIN(2.56) gives:

.549355436427

Remarks The argument is expressed in radians.

To convert to degress then:

J = SIN(A/180*PI)

SOUND

5.20.5 **Function** **SOUND** generates a note of a specified frequency and length.

Syntax SOUND pitch,duration.

Example SOUND 121,530

This will produce a middle 'C' note for one second.

The following program will play a short tune:

```
10 FOR I=1 TO 20
20 SOUND 52,1233/I
30 SOUND 46,1394/I
40 SOUND 59,1086/I
50 SOUND 121,530/I
60 SOUND 80,1202/I
70 FOR J=I TO 16
80 NEXT J
90 NEXT I
99 END
```

Remark Pitch and duration must be in the range 1 - 65,534.

A higher value for pitch will produce a lower note.

A higher value for duration will increase the length of the note.

The formula, 64130/pitch, can be used to find the required 'duration' parameter to play a note of a specified pitch for one second.

See section 9.12, FOUR OCTAVE SOUND RANGE, for parameters to generate specific notes of required frequency.

SPACE$

5.20.6 **Function** **SPACE$** is used to return a string consisting of n spaces.

Syntax SPACE$(n)

Example

```
10 DIM A$ (0,10)
20 A$ = "123" + SPACE$(3) + "ABCD"
30 PRINT A$
```

This will print a string consisting of the digits 1,2,3 followed by 3 spaces followed by the letters A,B,C,D.

Remarks The parameter n is an integer in the range 0 to 255.

SPC

5.20.7 **Function** SPC moves the cursor along the line by printing spaces

Syntax PRINT SPC(n) 0<n<255

Examples 100 PRINT"HELLO";SPC(10);"HOW ARE YOU ?"

Will display:

HELLO HOW ARE YOU ?

Remarks SPC must be used with one of the PRINT statements.

SQR

5.20.8 **Function** SQR returns the square root of the argument.

Syntax SQR(N)

where N is a numeric expression.

Examples Q = SQR(16)

sets Q equal to 4.

Remarks The argument must be positive.

SRCH

5.20.9 **Function** **SRCH** function enables an array to be searched for the occurrence of a specified element within that array.

Syntax SRCH(<array element>,<numerical expression>)

The array element specified is used as the reference to be searched for in the remaining array.

Examples
```
PRINT SRCH(A$)
A=SRCH(A(5),3)
IF SRCH(A!(0),1)<>5 THEN OPCHR7
```

Remarks Consider the following program.

```
 10 DIM A$(10,20)
 20 A$(0)="ABCDE"
 30 A$(1)="AB"
 40 A$(2)="ABCDE"
 50 A$(3)="ABCDE"
 60 A$(4)="ZXYZ"
 70 A$(5)="ABCDE"
100 INPUT X,Y
110 PRINT SRCH(A$(X),Y)
120 GOTO100
```

When found the element number is returned. If not found then zero is returned.

The function searches from the specified element to the end of the array. As an option, up to 255 occurrences can be checked for by specifying in the argument the number of repeat occurrences.

If this value is 256 or larger a syntax error will occur.

The result of SRCH is a numeric value which can be used in arithmetic expressions etc.

The search function can also be used on simple or double precision arrays.

The following are the results of search on the above array:

SRCH(A$) gives 2 : as element (2) is the first

SRCH(A$(2),1) gives 3 : as element (3) is the first occurrence of element (2)

SRCH(A$(4)) gives 0 : as element (4) is not found elsewhere.

When search is applied to numeric arrays the comparison must be exact for equality.

For string arrays the rules apply as in the 'IF' statement, i.e. the comparison is equal if the reference string is equal to, or a subset of, the searched string.
Example:

```
PRINT SRCH(A$(1))
```

would return 2 as A$(1) is a subset of A$(2).

STOP

5.20.10 **Function** **STOP** terminates program execution and prints a message on the LCD screen.

Syntax STOP

Examples 550 STOP

causes:

BREAK IN LINE 550

Remarks Typing **CONT** causes execution to continue from the line following **STOP**

STR$

5.20.11 **Function** **STR$** converts a numeric expression to a string.

Syntax STR$(N)

where N is a numeric expression.

Examples
```
10 A = 123.6
20 B$ = STR$(A)
```

Sets B$ = "123.6"

```
10 A = 7.941E12
20 B$ = STR$(A)
```

Sets B$ = "7.941E12"

Remarks This function permits the use of string operation on numbers.

Exponential notation numbers are represented literally in string form.

STRING$

5.20.12 **Function** **STRING$** returns a string of a given length composed of similiar characters.

Syntax A$ = STRING$(X,Y)

or

A$ = STRING$(X,Y$)

The first version will return the string A$ with x characters of ASCII code Y. The second version returns with a string of the first character of Y$.

Examples
```
100 PRINT"PAGE HEADING"
110 PRINT STRING$(12,"-")
```

Will output:

```
PAGE HEADING
------------
```

Remarks The following statements are equivalent:

A$ = STRING$ (5,65)

B$ = STRING$ (5,"A")

producing a string of five A's in A$ and B$ respectively.

SWAP

5.20.13 **Function** **SWAP** exchanges the values of two variables.

Syntax SWAP variable 1, variable 2

Example

SWAP A,B	this will put the value of variable A into B and the value of B into A.
SWAP A$,B$	this will put the contents of A$ into B$ and the contents of B$ into A$.

Remark Any type of variables may be swapped (single precision, double precision or string), but the two variables must be of the same type or a Type Mismatch error will occur.

TAB

5.21.1 **Function** **TAB** moves the print position cursor the specified number of places from the left-hand margin.

Syntax TAB(N)

where N is the PRINT POSITION to begin PRINTING.

Examples 100 PRINT"COL 1";TAB(10);"COL 2";TAB(20);"COL 3"

Will print:

COL 1 COL 2 COL3

Remarks In an LPRINT statement TAB specifies how many columns to skip before starting to print.

If the value of the numeric expression is less than the print position, then TAB is igrored. The value of expression must be positive and not exceed 255.

TAN

5.21.2 **Function** **TAN** returns the tangent of an angle.

Syntax TAN(N)

where N is a numeric variable in radians.

Examples T = TAN(2.56)

sets T equal to the value of the tangent of 2.56 radians.

PRINT TAN(2.56) gives:

-.65744712167262

Remarks The argument is expressed in radians.

The tangents on angles in degrees may be calculated as follows:-

A = TAN(angle/180*PI)

TIME$

5.21.3 **Function** **TIME$** returns an eight character string with the time. It is used for reading and setting the software independent calandar clock.

Syntax A$ = TIME$

OR

TIME$ = A$

Examples
```
10 DIM A$(0,8)
20 PRINT TIME$
30 A$ = "10:5"
40 TIME$ = A$
```

Remarks The eight character string is in the format hh:mm:ss. Leading zeros may be omitted, but, for example, if the minutes are zero, then one zero has to be entered.

To enter the time directly type the following:

TIME$="10:27"

Seconds must not be entered as this will give a Syntax error. Seconds are set to "00" when setting the clock.

TRON/TROFF

5.21.4 **Function** **TRON** and **TROFF** traces the execution of program lines by displaying their line numbers.

Syntax TRON

TROFF

Example
```
TRON
RUN
[10] [20] [30].....
.
.
.
.
TROFF
```

Remarks Line numbers are displayed in square brackets, but apart from this the program runs normally. The path of the program execution can be followed. TROFF turns off the trace facility.

VAL

5.23.1 **Function** **VAL** performs the inverse of STR$. It converts a string of digits into a number.

Syntax VAL ("STRING")

Examples
```
10 A$ = "123.6"
20 B = VAL(A$)
```
sets B to 123.6

Remarks Exponential notation numbers are converted literally.

Example:

```
10 A$ = "2.91E6"
20 B  = VAL(A$)
```
sets B = 2.91E6

VARPTR

5.23.2 **Function** **VARPTR** returns the address of the specified variable.

Syntax A = VARPTR(variable)

Examples
```
100 PRINT VARPTR(B)
```
This will display the address of variable 'B'.

Remarks VARPTR will only search RAM page RO.

WAND

5.24.1 **Function** **WAND** selects the bar code decoder to use.

Syntax WAND=n

Example WAND = 0

selects the CODE 39 decode software.

Remarks HUNTER can have multiple bar code decoder packages loaded. This command selects the active decoder.

Currently implemented are:

0 - Code 39
1 - EAN 8/13

NOTE: It is essential to have the relevant decoder software loaded in the factory prior to attempting to use a wand. See section 8.5 BAR CODES AND LIGHT PENS for further information.

WHILE..WEND

5.24.2 **Function** **WHILE....WEND** executes a set of statements if a condition is true.

Syntax
```
WHILE numeric expression
.
.
.
.
statements
.
.
.
.
WEND
```

'numeric expression' is evaluated true or false; if the expression is true the statements following are executed. If false, the statement after WEND is executed. The loops may be nested; each WEND associated with the most recent WHILE.

Examples
```
 10 FOR Q=1 TO 10
 20 B(Q)=Q
 30 NEXT
110 WHILE A< 11
120 FOR I = 1 TO 10
130 PRINT B(I);
140 NEXT
150 PRINT A:A = A+1
160 WEND
```

The matrix B is printed as long as the A parameter is less than 11.

Remarks Each time WEND is encountered the expression is re-evaluated. The same precautions should be exercised in jumping out of this loop as with FOR....NEXT loops.

NOTE: WHILE cannot be used with a string expression. The statements WHILE and WEND must be on separate lines and no other statements must be on those lines.

WINCHR

5.24.3 **Function** **WINCHR** inputs a single character from the optical wand.

Syntax WINCHR N

where N is a numeric variable.

Examples 100 WINCHR N
110 PRINT N

This will input a single character from the bar-code wand and display it on the LCD screen.

Remarks This function acts exactly like INCHR, but with the optional bar-code wand. The value is returned from the bar-code exactly like a keyboard input.

If a multiple character bar code is scanned then WINCHR will return the first character and the rest of the code will be ignored.

As with WINPUT if any key is pressed prior to the bar code input, then the input is taken from the keyboard instead.

NOTE: The bar code format must first be selected using the WAND verb. The relevant wand firmware must be resident in HUNTER as a system file. If it is not, then a system file error will result.

WINPUT

5.24.4 **Function** **WINPUT** acts exactly like INPUT, but with the optional bar-code wand.

Syntax WINPUT prompt, variable. (See also INPUT)

Examples 100 WINPUT A$
110 PRINT A$

The bar-code wand input will be stored in A$ and displayed on the LCD screen.

Remarks The string returned from the bar-code is read exactly like a keyboard input. If any key is pressed prior to bar-code input then the input will be taken from the keyboard instead.

Further details of available bar codes may be found in section 8.5, BAR CODES AND LIGHT PENS.

The use of string variables is recommended to allow for any alpha content.

NOTE: The bar code format must first be selected using the WAND verb. The relevant wand software must be resident in HUNTER as a system file. If it is not, then a system file error will result.

WRITE#

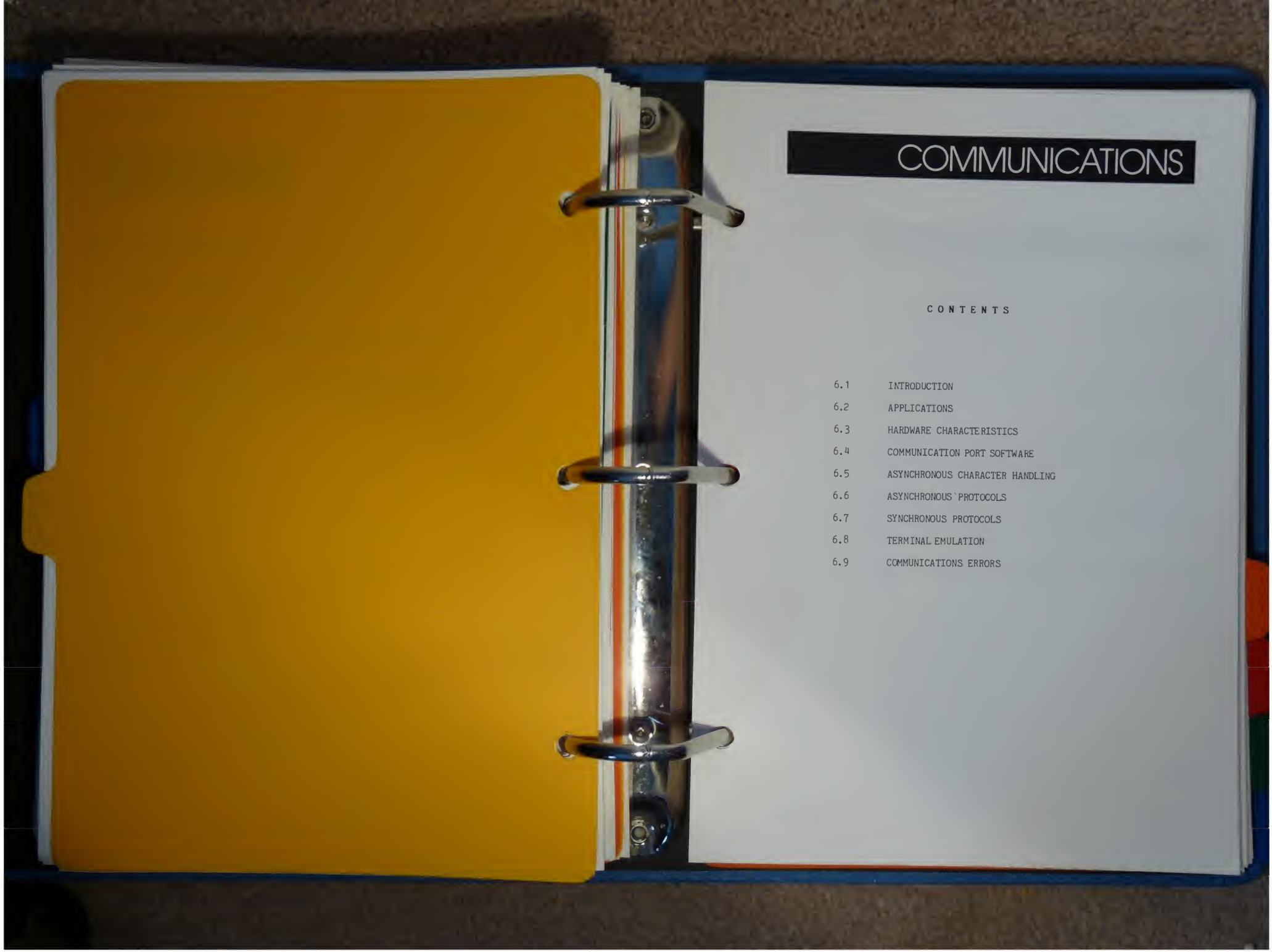
5.24.5 **Function** **WRITE#** outputs data to file

Syntax WRITE#file no,var1,var2,....varX

Examples WRITE#2,A$,B$,C

Will write the variables A$,B$ and C to file number 2.

Remarks The difference beteen WRITE# and PRINT# is that WRITE# will insert commas in the file between the items as they are written, while strings are delimited with quotation marks. WRITE# will not insert a blank before a positive number is written.


COMMUNICATIONS
CONTENTS
6.1 INTRODUCTION
6.2 APPLICATIONS
6.3 HARDWARE CHARACTERISTICS
6.4 COMMUNICATION PORT SOFTWARE
6.5 ASYNCHRONOUS CHARACTER HANDLING
6.6 ASYNCHRONOUS PROTOCOLS
6.7 SYNCHRONOUS PROTOCOLS
6.8 TERMINAL EMULATION
6.9 COMMUNICATIONS ERRORS

INTRODUCTION

6.1

A principle feature of HUNTER's architecture is its powerful and flexible communications facility.

HUNTER can communicate freely with most other types of computer, from micros to mainframes, using a variety of easily selected communication 'Protocols'. These protocols, of both synchronous and asynchronous varieties, are chosen for compatibility with other systems and are not special to HUNTER.

To establish communications, HUNTER uses the universally accepted RS-232/V24 communications interface. This interface, used in virtually every computer system, terminal, printer and modem, allows HUNTER to 'plug in' to other systems without modification.

HUNTER is equipped to act as a 'Data Terminal Equipment' (DTE), and together with appropriate protocol selections can resemble (partially emulate) many popular computer terminals. In this mode, it can be used as a portable computer terminal, accessing and communicating with other systems.

Alternatively, HUNTER can act as a self-contained computer, directly supporting peripherals like printers without additional equipment. Remember that in this mode, HUNTER may need a 'crossed cable' or 'modem simulator' to communicate with other terminal-equipped devices.

HUNTER's can communicate with each other, either side-by-side on the bench or over thousands of miles by telecommunication channels.

APPLICATIONS

6.2

HUNTER's communications can be used for transferring both programs and data to other systems.

Programs can be loaded into HUNTER daily, or as required, from a central database. Secure protocols and program-based error checks can guarantee accurate copies.

New, revised or corrected programs can be returned to the data-base for storage. Database files and command functions can be accessed automatically, or under manual control using HUNTER's terminal emulation.

Data can be collected and down-loaded to mainframe, minicomputer or microprocessor storage whenever required. Data transmissions can be formatted exactly to resemble existing data terminal configurations, allowing HUNTER to slot directly into existing software support structures.

Database information (for instance, tables of names and addresses) can be sent to HUNTER for presentation to operatives in the field. Collected data can be linked to previously loaded database information for instant verification and validation.

HARDWARE CHARACTERISTICS

6.3.1 **INTERFACE CONNECTIONS**

HUNTER uses a standard communications line via a 25 pin "D" connector compatible with the requirements of EIA specification RS-232-C. The connections may be summarised as follows in TABLE 6.1, EIA RS-232-C CONNECTOR SIGNALS.

TABLE 6.1 EIA RS-232-C CONNECTOR SIGNALS.

HUNTER is configured as a 'DTE' Data Terminal Equipment.

Pin No.	Description
1	Protective Ground
2	Transmitted Data
3	Received Data
4	Request to Send
5	Clear to Send
6	Data Set Ready
7	Signal Ground
8	Data Carrier Detect
9	(Not used)
10	5v Power Out
11	(Not used)
12	(Not used)
13	(Not used)
14	(Not used)
15	Transmit Clock
16	(Not used)
17	Receive Clock
18	(Not used)
19	(Not used)
20	Data Terminal Ready
21	(Not used)
22	Ring Indicator
23	(Not used)
24	(Not used)
25	(Not used)

6.3.2 **ELECTRICAL CHARACTERISTICS**

HUNTER's output voltages on signals "from HUNTER" are guaranteed to be greater in magnitude then 5V when driving a 4K7 load, in accordance with EIA specifications. Typically output voltages are 8V in magnitude.

HUNTER detects input signals with a "fail safe" characteristic. Input signals from -25V to +2.5V are detected as a logical "1" or the mark state. Signals from +3.5V to +25V are logical "0" or the space state. On the handshake signals the asserted state is spacing or logical "0" e.g. CTS.

To summarise,

Output:

Logical "0" or spacing	Output > +5V with 4K7 load.
Logical "1" or marking	Output < -5V with 4K7 load.

Input:

Logical "0" or spacing	Input > +3.5V
Logical "1" or marking	Input < +1.5V

Each input provides a line terminator impedance of 6K8.

6.3.3 INTERFACE POWER CONTROL

Under normal circumstances the serial output interface is powered down to conserve battery life. The first character which requires transmission causes the interface to be powered up. A pause of approximately 2 seconds is allowed for the interface to settle down and then the character is sent. Further characters are not affected in this way. Once powered the interface is latched on until HUNTER is switched off or power removed using an OUT instruction in either Basic or machine code.

When the interface is powered up it may cause spurious characters to appear to be sent out, causing problems with the receiving system. These may be overcome by powering the interface, prior to requiring the data, by sending any character as soon as HUNTER is switched on. A Basic LPRINT command will achieve this.

Alternatively, the interface may be controlled as follows:

```
OUT 132,1
```

will turn the interface on and

```
OUT 132,0
```

will turn the interface off.

If the interface is turned off after characters have been sent, then further characters will not cause the interface to be powered up and they will be lost.

6.3.4 HANDSHAKE LINES

The function of each of the handshake lines is described in turn in the following section, with the exception of Ring Indicator (RI). RI is provided to allow HUNTER to be switched on from an external source.

COMMUNICATION PORT SOFTWARE

6.4 HUNTER's communications format is **software controlled.**

If HUNTER fails to communicate, always check the port initialisation for the correct format.

6.4.1 **INITIALISING THE PORT**

To initialise, or alter, HUNTER's communications parameters as follows:

From **File Manager**, 'COMS', or application program select 'Initialise communications'.

You are now in the communications program. All entries are menu-type choices, selected by the cursor control keys and confirmed by 'enter'.

Values previously selected are displayed as the initial menu option presented. For instance, if '1200 baud' had previously been selected, then this option will appear after the heading 'Rate'.

HUNTER will prompt first for 'transmission parameters', followed by 'Receiving Parameters'.

Specific parameters are modified by positioning the cursor with the cursor left, cursor right keys, and selecting the desired option with the cursor up, cursor down keys.

NOTE: If the HUNTER has been totally reset for any reason, the parameters shown on entry to this program may differ from that expected. Extended use of the cursor keys will always correct any anomolies.

6.4.2 **TRANSMISSION PARAMETER SCREEN**

```
Transmission Parameters

Rate-1800   Prtcl-none    Pty-none

CTS-n DTR-n  LF-n Echo-y  T/O-no  Null-0

press ENTER if acceptable
```


6.4.3 **RECEIVING PARAMETER SCREEN**

```
Receiving Parameters

Rate-1200    Prtcl-none    Pty-none

RTS-off DSR-n  DCD-n  T/O-no  Serig-off

press ENTER if acceptable
```

To re-select the previous entries, press 'ENTER'.

6.4.4 **BAUD RATE (Rate)**

The available baud rates are:-

	1200
75	1800
110	2400
150	4800
300	EXT
600	

The EXT feature is to provide for external clocking when using synchronous protocols.

6.4.5 **SELECT PROTOCOL (Prtcl)**

The protocols available on HUNTER are:-

NONE	Systime
XON/XOFF	IBM 2780 (optional)
ACK/NAK	
ETX/ACK	

6.4.6 **SELECT PARITY (Pty)**

On transmission parity is implemented as follows:-

a) None — bit 8 is reset, i.e. logical 0.
b) Odd — bit 8 is set or reset to cause an odd number of logical 1's in the data bits.
c) Even — bit 8 is set or reset to cause an even number of logical 1's in the data bits.

d) 8-bit — All eight bits are transmitted with no modification. This may be used to transfer binary information.

On reception the implementation is:-

a) None — Parity bit is ignored and set to logical 0.
b) Odd or Even — The data is checked for an odd or even number of logical 1's. If the check fails then the character is received as 255 decimal (OFFH).
c) 8-bit — All eight bits of data are passed to the calling program.

NOTE: The parity fail character (OFFH) is displayed as ←-- on the screen.

6.4.7 HANDSHAKING LINES

There are handshaking lines provided on the RS-232 port. These signals are usually used when operating with a modem. The lines conform to the standard RS-232 specification for use with modems, for specific applications lines need to be enabled as appropriate. In general, a straight 25 way cable will operate with most modems. They may also be used when communicating with slow devices, e.g. printers or for the HUNTER to hold up incoming data. These lines will be discussed in turn.

6.4.7.1 **CTS** pin 5 (Clear to send) input to HUNTER.

This signal, selected on the transmit menu, controls data being transmitted.

If selected (y) then only if the line is active (+ve) will data be transmitted. The input is tested at the start of each character in async or start of each block in sync. Data characters are always completely transmitted even if CTS goes inactive.

If deselected (n) then this input is ignored and data is always sent.

6.4.7.2 **RTS** pin 4 (Request to Send) output from HUNTER.

The RTS handshake can be set to operate in three different ways from either the Communications Menu or via Basic by poking the RTS flag, see section 9.7, MEMORY LOCATIONS. The three modes are:

a) OFF — When this option for RTS is selected the RTS line is permanently active (+ve) and its state is not altered at any time during communications.

b) HOLD — In this mode the RTS line is normally active (+ve) and only goes inactive (-ve) when HUNTER has received too many characters and the receive buffer is about to overflow. This mode can thus be used to hold up a transmitting device until HUNTER has removed enough characters from the receive buffer such that there is enough space in the buffer to allow the transmitting device to send more data by raising RTS to active (+ve) again. This mode can be used when communicating HUNTER to HUNTER via a crossed cable. If a synchronous protocol has been selected, RTS should not be set to HOLD.

c) TRUE — This mode of RTS is meant for handshaking with a MODEM or similar device. When HUNTER wishes to transmit data, it first of all raises RTS from inactive (-ve) to active (+ve). Then HUNTER waits for a reply on the CTS line (if enabled) before sending the data in the transmit buffer. RTS remains active until all the data has been sent and the buffer is empty, then HUNTER returns RTS to the inactive state (-ve).

6.4.7.3 **DCD** pin 8 (Data Carrier Detect) input to HUNTER.

This signal, selected on the receive menu, is used to tell HUNTER that received data is valid.

If selected then only when the signal is active (+ve) is data received normally.

If DCD is inactive then no data will be received, even if the data input line has data on it.

6.4.7.4 **DSR** pin 6 (Data Set Ready) input to HUNTER.

This signal, selected on the receive menu, is used to tell HUNTER if the peripheral (usually a modem) is ready to operate. If selected, only if DSR is active will HUNTER communicate in both directions.

If DSR is inactive then HUNTER will not transmit or receive data. This signal can be ignored by selecting "n" on the receive menu.

6.4.7.5 **DTR pin 20 (Data Terminal Ready) Ouptut from HUNTER**

This signal, selected on the transmit menu, is provided to enable modems that require DTR to be present before generating carrier.

If selected (DTR-y), DTR will go to the active (+ve) state if either transmission or reception is attempted from the HUNTER.

If not selected (DTR-n), DTR will only be active if transmission from the HUNTER is attempted.

DTR can also be controlled by the communications power supply port 132 (see section 9.8), INVCON. DTR is active when the inverter is on.

6.4.8 **Timeouts**

Automatic communication timeouts are provided to signal to user programs that the communications lines are quiescent. Separate timeouts are user-selectable for both transmit and receive channels. The timeouts are designed for use with the Basic ONCOMMS statement (see section 5.16.3), but can also be invoked manually or by machine code programs. The error number is available in the location COMERR (see section 9.7, page 9-38).

The timeouts have a range of 10 - 60 seconds in 10 second steps and are controlled by memory locations RXTOAF and TXTOAF (see section 9.7, page 9-38). These locations can be read or written to by user programs. When set to 0 (cleared), the timeouts have no effect. When set, they have the following effect:

6.4.8.1 **RXTOAF**

Will cause a 'communications error' if the receive line is quiescent for more than the selected period. Reception of any character, or noise, will reset the timeout. The timeout is initiated by a request for serial input data through a Basic 'LINPUT' or similar statement. If no attempt is made to read the input communications buffer, the timeout will not occur.

6.4.8.2 **TXTOAF**

Will cause a 'communications error' if the HUNTER is inhibited by the hardware handshake lines CTS (Clear to send) or DSR (Date set ready). If either line inhibits transmission from the HUNTER, a 'communications error' will occur after the specified time.

6.4.9 **Echo**

The 'echo flag', ECHOF (see section 9.7, page 9-37) determines whether transmitted characters are echoed on the HUNTER screen. Data transmission and reception are not affected and the displayed characters are not passed to user programs or entered in the receive buffer.

Echo can be regarded as a full/half-duplex selection, with duplicate characters appearing on the screen if the HUNTER is communicating with a full-duplex host with echo 'on'.

6.4.10 **LF**

The line feed flag, LFACT (see section 9.7, page 9-37) determines whether a LF character (decimal 10, Hex '0A') is inserted in the data stream transmitted by HUNTER automatically following every CR (Decimal 13, Hex '0D') in the transmitted text.

If selected, on (LF-y), line feeds are inserted after CR even if line feed characters are already present in the data.

If selected, off (LF-n), line feeds are not inserted. However, line feeds present in the data will be transmited.

NOTE: When sending binary data, the LF-y option will insert LF characters if the data contains decimal 10, even if the decimal 10 character is not intended as a CR. Always select LF off (LF-n) if using 8-bit transmission.

6.4.11 **Null**

The Null count is provided in the transmission set up for use with printers that have a significant carriage return delay and when no handshaking or buffer is available.

If selected, NULL inserts NUL (Decimal 00) characters in the data stream after CR (and LF, if selected with the LF-y option).

The number of nulls is chosen from 0,2,5,10 or 20. these options are provided by the HUNTER operating system and cannot be varied.

NOTE: If transmitting binary files in 8 bit mode, null must be selected off (Null-0), since it will otherwise insert Null (Decimal 0) characters after every CR (Decimal 13) character.

6.4.12 **SERIG (Receive channel only)**

The SERIG location (see section 9.7, page 9-38) allows any single character in the range 0 - 127 Decimal ('0' - '7F' Hex) to be ignored by HUNTER's serial data reception system.

Characters rejected by SERIG are not entered in the communications buffer or passed to user programs.

SERIG is provided for use with computer systems that provide abundant spurious characters (e.g. XON or DEL) as part of data streams. Since filtering these unwanted characters can be inconvenient in user programs, SERIG provides a useful solution.

The value of SERIG is selected by incrementing/decrementing the decimal value using the up/down cursor keys and **must** be set to 'OFF' when not in use. Serig-off is obtained by incrementing past 127 or below 0.

ASYNCHRONOUS CHARACTER HANDLING

6.5.1 HUNTER's asynchronous serial data communications are handled by a communications software package totally separate to the user's program or the Basic interpreter.

The user needs no detailed knowledge of data communication to be able to utilise HUNTER's powerful facilities. The required configuration is selected from 'menus' and operation of the package is virtually invisible.

6.5.2 **BUFFERS**

Characters are transferred to and from the serial interface via invisible 'buffers' that effectively disconnect the application program from the outside world. These buffers ensure a smooth flow of data and correct reaction to protocol responses even when HUNTER is busy doing something quite different!

Because the communication package is interrupt driven, the execution of user programs is not affected by communication. However, the user program **may slow down** if the serial line is busy. HUNTER's communication buffers are available whenever the machine is powered up, even if the application program is **not requesting communication.** Incoming messages will be received and held in the buffer, while any outgoing data remaining will continue to be sent down the line. Any protocols selected will continue to be observed.

6.5.3 **COMMUNICATING WITH USER PROGRAMS**

A user program can transmit and receive characters either singly or in blocks. The program may be written in HUNTER BASIC (see LPRINT, LINCHR, etc) or a compiled language using the standard CP/M calls.

Irrespective of the content of the user program, all data rates, protocols and other selections previously made will be observed invisibly by HUNTER's communication software package.

6.5.3.1 **Transmission**

When transmitting, HUNTER writes the characters into a buffer. If the buffer becomes full, then the user program (BASIC or CP/M) is held up until the characters or block are sent.

If a 'communication failure' is detected, execution of the user program will be suspended until manually overridden.

6.5.3.2 **Receiving**

When **receiving** characters, data receiving commands (LINCHR and LINPUT in Basic) will fetch any characters or blocks currently held in the reception buffer.

NOTE: This data may have been received **prior** to execution of the input command

If data is already in the buffer, then it will be returned immediately. If no data is present, HUNTER will wait for incoming data, and will return when characters or blocks are available.

Reception of data is completely transparent to the user Basic program being run by HUNTER. Data may be sent at any time to HUNTER, communications protocols being implemented as selected.

Use is made of a 256 character input buffer. Received characters are placed into the buffer in a 'barrel' fashion. If more characters are received than this, without any being read by a calling program then the earliest characters are overwritten. This is true even if RTS or XON/XOFF handshaking is used and is ignored by the sending device.

The effect of this buffering is to permit reception of characters to carry on steadily through calling programs which process the characters intermittently. This can give a greater overall throughput.

The buffer is needed by definition for the XON/XOFF and ETX/ACK protocols.

ASYNCHRONOUS PROTOCOLS

6.6 **NOTE:** In this section, 'host' means the device communicating with HUNTER, whether it be computer, printer, modem or otherwise.

These protocols may be used by Basic application programs, user assembly code, CP/M programs or directly in 'Terminal Emulation' mode.

6.6.1 **NONE**

RECEPTION
As implied, this selection does nothing to control the incoming data. It behaves as a simple teletype. Characters are placed into the reception buffer and a calling program may read them out asynchronously.

TRANSMISSION
Each character is transmitted as the calling program requests it (except when waiting for the previous character to be sent). There is no buffering of the output data.

6.6.2 **XON/XOFF (DC1/DC3)**

RECEPTION
If an incoming character causes the receive buffer to have 90 or more unread characters then an XOFF character (DC3,19 Decimal or Control S) is transmitted, which should prevent the host sending further characters to HUNTER. When the buffer is reduced to 10 pending characters, an XON (17 Decimal or Control Q, DC1) is transmitted to re-enable the host to HUNTER transmission. Subsequent XONs or XOFFs are never transmitted without an intervening XOFF or XON respectively.

TRANSMISSION
The input data line is also examined for XON or XOFF characters.

If an XOFF or stream of XOFFs are received then transmission is stopped as soon as the character being transmitted is finished. The protocol requires an XON to restart transmission. This protocol procedure strips 'XON' or 'XOFF' Control characters, thus returning only valid data.

6.6.3 **SLAVE, HUNTER TO HOST TRANSMISSION**

TRANSMISSION
It should be noted that following the host transmitting an XOFF, it is possible, depending on data link speed, to have a further two characters sent out by HUNTER. This is due to the time taken to receive the XOFF and the possibility of just missing

6.6.3 **SLAVE, HUNTER TO HOST TRANSMISSION**

TRANSMISSION
It should be noted that following the host transmitting an XOFF, it is possible, depending on data link speed, to have a further two characters sent out by HUNTER. This is due to the time taken to receive the XOFF and the possibility of just missing the start of the next character.

At the start of transmission a received XON is assumed, to avoid lock-up.

6.6.4 **ETX/ACK**

RECEPTION
In this protocol HUNTER will send an ACK character (06 Decimal or Control F) after reception of an ETX character terminating a line that has been ready by Basic. The ETX is stripped from incoming data and not sent to Basic. This protocol causes the host device to wait at the end of each block of data, which should not exceed 130 characters, for an acknowledgement that the block has been read. The ETX/ACK acknowledgement is only transmitted when the block has been fetched from the buffer by Basic.

TRANSMISSION
This protocol will send out an ETX (03 Decimal or Control C) character after any transmitted carriage return. Transmission is then halted until an ACK (06 Decimal or Control F) is received from the host. This enables individual data blocks to be sent and an acknowledgement awaited. If transmission is to another HUNTER the blocks should not exceed 130 characters or the input buffer may be exceeded. If an ACK is not received within thirty seconds then communication failure is assumed and an error message is put up on the screen.

The ACKs are not returned to the user program.

By the very nature of this protocol, indeterminate handshaking situations may occur if any conditions are altered during transmission.

6.6.5 **ACK/NAK**

RECEPTION
In this mode HUNTER will receive a complete block of data, terminated with a carriage return. It will then check for parity errors. If one exists then a NAK (21 Decimal or Control U) is sent out to the host and the data block ignored.

If the block is error-free then the received block is sent character by character to Basic. When the carriage return is sent to Basic an ACK (06 Decimal or Control F) is sent out to the host to indicate successful data block reception.

TRANSMISSION
Data blocks are first placed into a transmission buffer. Upon receipt of a carriage return from Basic, the entire buffer is transmitted. HUNTER then awaits an acknowledgement. If an ACK is received then the next data block is assembled into the buffer and sent. If a NAK (21 Decimal or Control U) is received then a failure is indicated and the entire buffer is sent again. If nothing is received after three seconds a NAK is assumed and the buffer sent again. The NAKs may be sent a maximum of three times. If an ACK is still not received then the communications failure message is put on the screen.

6.6.6 **SYSTIME**

6.6.6.1 **HUNTER to Systime communications Protocol Specification**

This protocol is provided to facilitate communication between HUNTER and Systime model S500 minicomputers. The protocol embodies a checksum in addition to parity checks.

Data is sent in the form of messages which are subdivided into one or more fixed length blocks. Each block is individually checked and acknowledged before any further blocks are sent.

DATA BLOCKS

Each block consists of:-

1 start character
64 data characters
3 checksum
1 stop character

The purpose of each character group is as follows:-

Start Character
The first block of a message uses SOH (01 Decimal). Further blocks use STX (02H) as the start character.

Stop Character
The last block of a message uses EOT (04 Decimal), previous blocks use ETX (03H) as the stop character.

Data Character
The data characters may consist of any 7 bit pattern with the exception of the handshake characters : ACK (06 Decimal), NAK (15H) and WACK (13H).

Checksum
The checksum is formed as the modulo 256 addition of all data characters, with parity reset. The number formed is transmitted in ASCII as three decimal digits. Leading zeros are not suppressed. The most significant digit is sent first.

Insufficient Data Characters
If there are less than 64 characters for the data block, then the remainder of the block is filled with NULLS (00 Decimal). The characters are ignored on reception.

Handshaking
Each block of data must be acknowledged by the receiving device. After a successful reception and if the receiver is ready to receive another block, then it returns an ACK (06 Decimal). If the transfer is deemed unsuccessful for any of the following reasons:-

1) parity fault on any character
2) Incorrect start or end characters
3) wrong number of characters
4) checksum error
5) timeout (waiting for characters for 4 secs)

then a failure is returned as a NAK (21 Decimal).

If the receiver accepts the block, but is not ready for further data, then a wait acknowledge WACK (19 Decimal) character delaying the transmitter's time out is transmitted. Further delays can be accumulated by transmitting WACK's at the rate of one every 2 secs. The message is finally accepted with an ACK.

6.6.7 COMMUNICATIONS FAILURE

If a failure occurs on transmission and a timeout activates, then HUNTER will display 'Communications Failure' on the top line of the display. This message may be overriden by pressing 'X' on the keyboard. This will force transmission for ETX/ACK or XON/XOFF; in the case of ACK/NAK then the buffer will be sent another three times.

A timeout can be prevented from occurring after a transmission failure by re-initialising the communications.

SYNCHRONOUS PROTOCOLS

6.7

To provide the fast, high integrity communication required by most mainframe computers, HUNTER has a synchronous data communication capability.

Communication takes place using synchronous transmission of data blocks. These blocks are checked using check characters and acknowledged as appropriate using particular protocols.

Currently available as a factory option is the IBM 2780* communication protocol providing the bisynchronous communications. Other protocols are in development.

6.7.1 IBM 2780 ON HUNTER

BACKGROUND
The IBM 2780 is a very popular data transmission terminal used for remote job entry. The transmission protocol developed for this device has been widely implemented on other mainframes to the extent of becoming a de facto industry standard. HUNTER's implementation mimics 2780 protocol to provide well proven, fast and extremely reliable data transfers.

IBM 2780 is used for batch entry by use of "job cards". HUNTER is easily programmed to simulate the use of job cards providing all the processing facilities normally available on a remote terminal.

HUNTER is the most advanced portable to offer a comprehensive implementation of IBM 2780 in true synchronous mode.

ADVANTAGES
This well proven and sophisticated protocol gives HUNTER reliable, high speed communications over relatively poor data links such as the public switched telephone network at speeds as high as 2400 baud.

Use of this standard provides access to a wide range of mainframe computer systems, directly and without need for supporting equipment. Data files can be exchanged between HUNTER and mainframe with great flexibility, convenience and dependability.

*NOTE: IBM and 2780 are registered trademarks of IBM Corp.

TABLE 6.2 TECHINCAL IMPLEMENTATION

: Synchronous transmission using Bisync.
: Character set - EBCDIC
: Full CRC generation, error handling and flow control.
: Modem handshake lines available.
: Internal or external clocking options.
: May be used over the public switched telephone network or private lines.
: Records terminated using the EM character, providing optimum throughput.
: The "Wait acknowledge" is fully implemented.
: Protocol use is completely transparent to user programs.
: Data Rates - up to 9600 Hz with external clocking.
- up to 4800 Hz using internal clocking

The internal self-synchronising feature enables HUNTER to HUNTER communications over standard 300 baud asynchronous modems.

6.7.2 SYNCHRONOUS COMMUNICATION

Synchronous communication transfers data as a continuous stream of data characters, without the need for start and stop bits as used in asynchronous communication. This maximises the data transfer rate for a given channel bandwidth. For the receiver to be able to use the data two levels of synchronisation need to take place:

i) **Bit Synchronisation**
This allows the receiver to save the state of each bit correctly, preferably sampled at the centre of each bit.

ii) **Character Synchronisation**
Each block of eight bits which represent a character needs to be identified from within the data stream.

6.7.2.1 **Bit Synchronisation**

Two techniques are used by HUNTER to achieve bit synchronisation.

External Clocking allows both transmit and receive to be synchronised using two independent clocking signals. These signals are generated by either a modem or modem eliminator. The selection 'CLK' on the speed selection of transmit or receive menus selects this mode.

Two pins are provided for the clock signals. Pin 15 provides the transmit clock and pin 17 the receive clock.

On transmit, a rising edge (-12V to +12V) causes a bit to change on the transmit output (pin 2). On receive, a falling edge (+12V to -12V) causes the state of the receive input (pin 3) to be sampled.

It is **important** that maximum clock speed on HUNTER should not exceed 9600 Hz (giving a data rate of 9600 baud). The minimum speed is approximately 100Hz.

Exceeding this clock rate may cause unpredictable results.

Alternatively, HUNTER can **self clock.** This is achieved by selecting from the menu the desired transmit and receive baud rates in the range 50-4800 baud, as for asynchronous. On transmit, the data is sent out by using internal timing, with no reference to any external clock source.

To receive incoming data HUNTER times from rising edges (-12V to +12V) on the receive data line to give internal clocking.

Self clocking is useful for HUNTER - HUNTER communications. It may be used over asynchronous modems to provide both increased throughput and data integrity.

6.7.2.2 **Character Synchronisation**

To be able to extract eight bit data characters, HUNTER has to synchronise itself to the incoming characters correctly. This is done by detecting special synchronising characters called SYN characters (32H). HUNTER considers synchronisation to have been achieved after receiving two contiguous SYN characters.

After receiving the two synchronising characters each group of eight received bits are then considered to be a character.

Synchronisation is considered to have been lost if a line turnaround character is received (CFFH). This is equivalent to an open line.

Prior to sending the synchronising characters, HUNTER transmits an OAAH pad character which is to assist in bit synchronisation.

Every data block whether it consists of one or 500 characters, must start with a pad character, and two SYN characters and terminate with line turnaround.

6.7.3 BINARY SYNCHRONOUS COMMUNICATIONS (BSC)

The information carrier for 2780 is the binary synchronous communications (BSC) procedure. The character set used is EBCDIC (Extended Binary Coded Decimal Interchange Code). The following is a description of the subset of BSC used for HUNTER 2780 emulation.

NOTE: HUNTER generally operates using the USACII (United States of America Standard Code for Information Interchange). Characters either transmitted or received by Basic will be in ASCII, so CHR$, LOPCHR, etc., instructions all generate ASCII characters. To use EBCDIC the 2780 driver software utilises an ASCII to EBCDIC and EBCDIC to ASCII code converter. The conversions used are listed in section 9.11, ASCII TO EBCDIC CONVERTION.

The code used on the transmission channel is transparent to Basic programs, and need not concern the programmer.

6.7.3.1 Text Blocking

BSC defines general structural procedures for the blocking of information. Several control characters are used for the control of message blocks.

A message consists of one or more blocks of text data. The start of text character (STX) is used immediately preceeding each block of data. Each data block but the last is terminated by an end of transmission block (ETB) character or an intermediate text block (ITB) character. The last data block ends with an end of text (ETX) character.

6.7.3.2 Error Checking

Each block of data is error checked by the receiver by a cyclic redundancy check (CRC). After each transmission the receiver normally replies with ACK0 or ACK1 - data accepted continue sending; of with NAK - data not accepted (e.g. due to a data transmission error), retransmit the previous block.

6.7.3.3 CRC-16

HUNTER uses the accepted error checking method of EBCDIC 2780. The cyclic redundancy check is a division at both the transmitting and receiving stations using the numeric value of the message as a dividend, divided by a constant. The quotient is discarded and the remainder is used as the check character.

This is transmitted immediately following an ITB, ETB or ETX.

The receiver checks this value and finds no error if they are equal.

CRC-16 uses the polynomial $(x^{16}+x^{15}+x^2+1)$ as the divisor constant. The 16 bit remainder is sent as two eight bit characters. This is also sometimes referred to as the BCC (Block Check Character).

6.7.3.4 Line Bid

For either HUNTER or host computer to transmit a message, the line must first be seized. This requires sending an enquire (ENQ) character. A positive acknowledgement permits the start of a message.

6.7.4 LINK CONTROL CHARACTERS

The data link is controlled by the use of the following control characters:

6.7.4.1 SYN (32H) - Synchronous Idle

This character is used to establish synchronisation. Two contiguous SYNs are required for synchronisation. They may also be used as pad characters, although never used as such by HUNTER. Syncs will be ignored after synchronisation has been established except as part of CRC.

6.7.4.2 STX (02H) - Start of Text

This character is used as the start of text preceeding all text blocks. It is not part of the CRC accumulation.

6.7.4.3 ETB (26H) - End of Transmission Block

The ETB character indicates the end of a block of data characters. A block check character is sent immediately following an ETB. A reply is required after an ETB (ACK0, NAK, etc).

6.7.4.4 ITB (1FH) - End of Intermediate Transmission Block

ITB is also called IUS or US (Unit Separator). This enhances the throughput due to not having to go through a line turnaround. The CRC accumulation is reset after an ITB. It is not necessary to send an STX after the ITB.

If the error check fails then a NAK is sent at the end of the transmitted message, i.e. after the ETB or ETX. The whole of the message is retransmitted in the event of a NAK being the reply to a complete message block.

6.7.4.5 **ETX (03H) - End of Text**

ETX is transmitted after a block of characters started by an STX. The CRC is sent immediately after an ETX. A reply is required to indicate the receiver status.

6.7.4.6 **EOT (37H) - End of Transmission**

This character terminates a message transmission containing a number of blocks. It is also used to indicate a system malfunction.

6.7.4.7 **ENQ (2DH) - Enquiry**

ENQ is used as the line bid to start a transmission sequence. ENQ is also used to obtain a repeat transmission of a response, in the case of a garbled response.

6.7.4.8 **ACKO (10H,70H) - Affirmative Acknowledgement**

This is one of two positive acknowledgements to error checked message buffers. This informs the transmitting station that the message was received satisfactorily. This acknowledgement is also used to confirm a request for the line. ACKO alternates with ACK1 positive acknowledge.

Alternating Affirmative Acknowledgement
Alternating ACKO and ACK1 provides sequential checking of replies. this ensures that the blocks are replied to correctly. ACKO is used as the response to a line bid.

It should be noted that ACKO and ACK1 consist of two 8 bit characters.

6.7.4.9 **ACK1 (10H,61H) - Positive Acknowledge**

This is the alternate acknowledge. It is used with ACKO to respond correctly to incoming messages.

6.7.4.10 **WACK (10H,1CH) - Wait Positive Acknowledge**

A receiving station that has correctly received a message but is not yet in a condition to use the data, can send a WACK. It is a positive acknowledge. The transmitting station should then

respond with an ENQ. WACK is also a two character sequence. WACKs may be sent indefinitely at 3 second intervals to stop further transmission.

6.7.4.11 **NAK (15H) - Negative Acknowledge**

This character informs the transmitting station that a message was received in error. It causes retransmission of the erroneous message.

6.7.4.12 **DLE (10) - Data Link Escape**

Data link escape is used to provide various line control characters, e.g. WACK, ACKO, ACK1 and RVI.

6.7.4.13 **RVI (10H,7CH) - Reverse Interrupt**

Reverse interrupt is used as a positive acknowledgement (in place of ACKO or ACK1). However, it requests termination of the current message, due to a high priority message needing to be sent from the receiving station.

HUNTER will never generate an RVI but will respond to one.

6.7.4.14 **DLE EOT (10H,04H) - Disconnect Sequence**

This sequence informs the receiver that the transmitter is shutting down.

6.7.4.15 **IRS (1EH) - Intermediate Record Separator**

This character is used as a delimiter in 3780 configurations. Reception of this returns a CR (carriage return).

6.7.5 **MESSAGE TRANSMISSION**

The following illustrate various message sequences and error handling.

NOTE: TX = Transmitting station. RX = Receiving station.

Fig 6.1 NORMAL MESSAGE

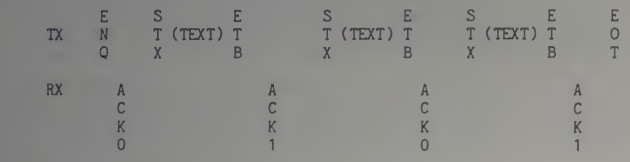
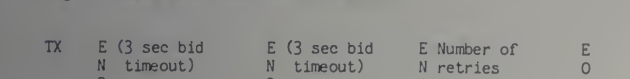


Fig 6.2 UNANSWERED LINE BID (ERROR 01)



NOTE: HUNTER will retry for a total of 10 ENQs. Many mainframe systems never give up.

Fig 6.3 ACCEPTED RE-TRANSMISSIONS

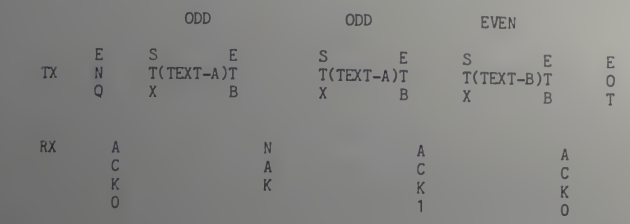
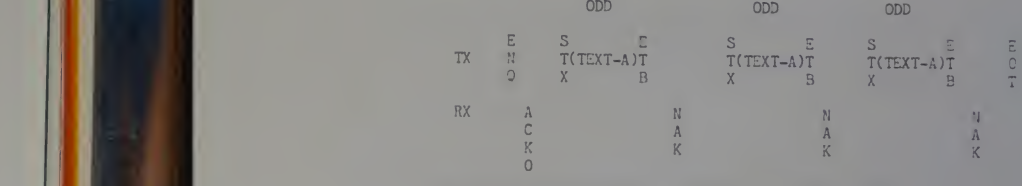
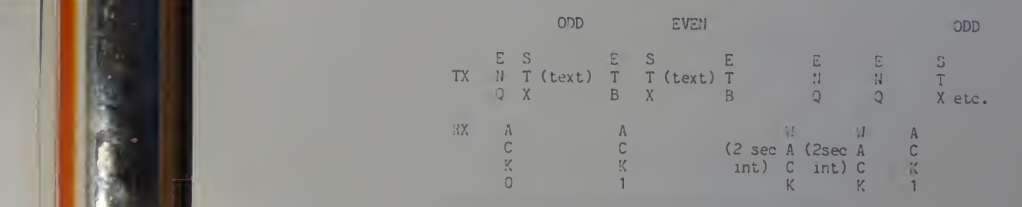


Fig 6.4 RE-TRANSMISSION REJECTED (ERROR 03)



NOTE: Number of re-trys can vary. HUNTER will retry 10 times, but some mainframe systems will retry indefinately.

Fig 6.5 TRANSMISSION DELAY (RECEIVER INITIATED)



NOTE: HUNTER does not count NACK-ENQ sequences. The receiver buffer may be cleared.

Fig 6.6 TRANSMISSION DELAY (TRANSMITTER INITIATED)

TX	ENQ	ST (TEXT) X	ET 2 SEC B	TT 2 SEC INTERVAL D	TT D	ST (TEXT) X	ET X		EOT
RX								ACK 0	
			ACK 1		NAK	NAK			

NOTE: HUNTER does not count TTD/NAK sequences or generate TTDs.

Fig 6.7 STX LOST AND DATA IGNORED

		ODD	EVEN			EVEN		
TX	ENQ	ST(TEXT A) X	ET(TEXT B) B	ET (3 SEC TIMEOUT)	NAK Q	ST(TEXT B) X	ET B		EOT
RX				NO RESPONSE				ACK 0	
			ACK 1			ACK 1			

Fig 6.8 INCORRECT POSITIVE ACKNOWLEDGEMENT (ERROR 03)

TX	ENQ	ST(TEXT A) X	ET(TEXT B) B	ET	ENQ	ENQ	ENQ		EOT
RX								ACK 0	
			ACK 1		ACK 0	ACK 0			

NOTE: HUNTER will send 10 ENQs before giving up.

Fig 6.9 DATA LINK ABORTION ON NO RESPONSE (ERROR 02)

TX	ENQ	ST (TEXT) X	ET (3 SEC RESPONSE B TIMEOUT)	ENQ (3 SEC TIME OUT)	ENQ			EOT
RX			NO RESPONSE	NO RESPONSE				
	ACK 0							

NOTE: HUNTER sends out 10 ENQs.

Fig 6.10 DATA LINK STALEMATE

TX	ENQ	ST (TEXT) X	ET B	NO CONTINUATION		
RX						
		ACK 0		ACK 1		

NOTE: Failure to receive further messages could be timed out by the Basic applications program.

6.7.6 HUNTER/2780 SPECIFIC IMPLEMENTION FEATURES

The foregoing describes in general operations on the communications line for 2780 protocol. HUNTER has particular features and handling which may be important to users.

6.7.6.1 Buffer Operation

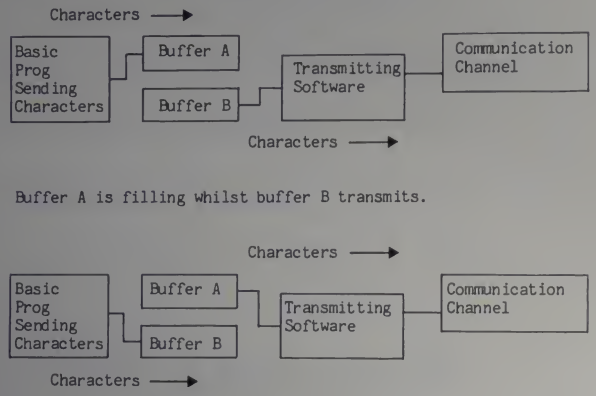
To implement this synchronous protocol it is necessary to use buffering on transmit and receive so that error checking and re-transmission can be performed.

To enhance throughput there are two buffers each for transmit and receive, the operation of which is described below.

6.7.6.1.1 **Transmit Buffer**

On transmit while one buffer is being filled from the user program the other is being sent. When the new buffer is full the buffers effectively swap over and the cycle is continued:

Fig 6.11 TRANSMIT BUFFER

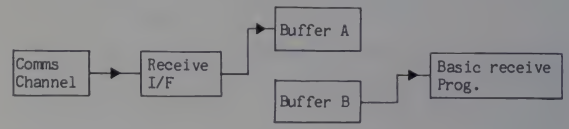


Buffer A is filling whilst buffer B transmits.

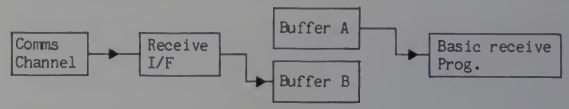
Buffer B is filling whilst Buffer A transmits.

The maximum buffer length, including control characters, that can be received is 512 characters. However, some systems are only configured for 256 characters. For this reason HUNTER will only default to transmitting 200 characters for ease of interfacing.

Fig 6.12 RECEIVE BUFFER



Filling buffer A, emptying buffer B.



Filling buffer B, emptying buffer A.

As with the transmit buffer arrangement, one buffer is being filled from the communications line while the other is being emptied by the user program.

6.7.6.2 **Data Format**

HUNTER is able to accommodate 2780 protocol as both fixed and variable block lengths. Also, 3780 is accommodated.

It is possible to configure the bisync emulator to provide very flexible interfacing for a variety of different host machines.

Standard configurations can be made for:

a) 2780 with variable record length
b) 2780 with fixed record length
c) 3780

The block sizes and record sizes can be adjusted for compatibility and maximum throughput on the communication channel.

Other features which are available for special purposes are:

a) Intermediate STX
b) Single record blocks
c) Space compression

Features not implemented are:

a) Transparency
b) Multi drop

All features are discussed in detail later. Although they may all be selected individually, they may not make a compatible protocol. Standard configurations are as follows:

6.7.6.2.1 **2780 with variable records**

(This is the default configuration)

a) Record length = 80
b) Block length = 200
c) Multiple records per block
d) Record Padding disabled
e) No intermediate STX

6.7.6.2.2 **2780 with fixed length records**

a) Record length = 80
b) Block length = 200
c) Multiple records per block
d) Record padding enabled
e) Intermediate STX enabled

6.7.6.2.3 **3780**

a) Record length = 80
b) Block length = 512
c) Multiple records per block
d) Record padding disabled
e) No intermediate STX

Space compression is only normally used in 3780 installations.

6.7.6.3 **HUNTER Card Format**

Virtually all HUNTER data is delimited by CR (carriage return).

Blocks of data loaded by LINPUT statements are terminated by CR, as are lines of printout. The most straightforward method of block creation is to treat data between CRs as cards of data.

6.7.6.3.1 **Transmitting Cards**

On transmit, if a CR is detected in the data, the communication software will treat it as a record end. The action taken depends upon the amount of room left in the current buffer being prepared for transmission.

For further details on end of record action see section 6.7.6.5.

6.7.6.3.2 **Sending the last Card**

To maintain compatibility with existing transmission programs it is essential they should operate without modification. This leaves a problem of how to send the final message buffer if it has not been filled up. If greater than 5 seconds elapses between characters sent from the Basic applications program, then the end of a complete transmission sequence is assured and the current block terminated by ETX CRC. After this has been acknowledged the EOT (end of transmission) is sent to close down the line.

6.7.6.3.3 **Receiving Cards**

The computer generating data for transmission to HUNTER will also block data into records.

To provide application programs with "CR" delimiters, any end of record is converted to CR. The end of record will depend upon the configuration in use. The sequences IUS CRC and IRS will always return CR.

This means that HUNTERS will communicate directly when connected back to back.

BSC receiving stations use ESC sequences for special functions like skipping line. These sequences are ESC n where n can be a number of printable characters. To avoid confusion when these are received HUNTER will ignore received ESCs and their associated following character.

6.7.6.4 **Character Restrictions**

HUNTER does not implement any of the transparency features of BSC. It is, therefore, not possible to send any of the following characters:

STX - 02H
ETX - 03H
ETB - 17H
EDT - 04H
SYN - 16H
NAK - 15H
DLE - 10H
US - 1FH
ESC - 1BH

If any of these characters are sent then they may either be lost or cause disruption to the transmission channel.

6.7.6.5 **Configuring the Protocol**

For maximum flexibility there are a number of features which may be controlled by the user. These are controlled by a number of flags which are memory locations, listed through this section and also in section 9.7 Memory Locations. All of these locations are preserved after timing set, as with all other communications parameters such as transmission speed.

They may be changed from within a user program using POKE in Basic or direct writing in machine code. Alternatively, the Basic program "SENTSYNC" will give a menu of options. A listing is included in section 9.5.

6.7.6.5.1 **Record Length - (Memory location F853H or 63571)**

This is a single byte which defines the maximum record length. It has a range of 1-255, with a default of 80. When outputing data, this length should not be exceeded as there may be insufficient space left in the block to accommodate the record.

If record padding is used, all records are either padded or truncated to fit this length.

6.7.6.5.2 **Buffer Length - (Memory locations F854-F855, 63572-63573)**

This is a two byte number which defines the maximum length of a block. It has a permitted range of 10-512 with a default of 200. The block length must be greater than the selected record length. This control is useful for adjusting the throughput for good or bad telephone lines when transmitting over modems or acoustic couplers. For a good line, the buffer length could be set to maximum so that line turnarounds are kept to a minimum and the maximum data is sent in one block. If, however, the line is a bad one, then if the buffer length is set to a single record length, the amount of data to be re-transmitted in the event of a NAK or data corruption due to the bad line, is kept to a minimum. On Receive, the block length can be anything up to 512 characters.

It should be remembered that the block includes all control characters and record separators.

6.7.6.5.3 **Single Record Flag - (Memory location F856H or 63574)**

This flag controls whether there is single or multiple records in a block. If set to zero (default value) there will be multiple records in each block. If set non-zero, there is only one record transmitted per block.

6.7.6.5.4 **Record Padding flag - (Memory location F857 or 63575)**

This single byte flag is used to control record padding. If this byte is set to non-zero, a record is padded with spaces to its maximum length as defined by record length before being transmitted. On receiving a record, trailing spaces are stripped off. If this flag is set to zero, record padding is inhibited and trailing spaces are not stripped on reception. The default state of this flag is zero.

Record padding is only normally used on fixed length 2780 communications.

6.7.6.5.5 **Mode flag - (Memory location F858 or 63576)**

This flag determines the essential difference between 2780 and 3780. It selects between terminating a record with US (IFH), followed by CRC, or with IRS (IEH) followed by CRC.

2780 mode is the default with the mode flag set to 0. A non-zero flag selects 3780.

2780 mode:

Fixed length records are terminated by any of the following:

US,CRC : used for a partly filled buffer
ETB,CRC : used for a full buffer
ETX,CRC : used for the last buffer to be sent

Variable length records are terminated by any of the following:

EM,US,CRC : used for a partly filled buffer
EM,ETB,CRC : used for a full buffer
EM,ETX,CRC : used for the last buffer to be sent

3780 mode:

All records are terminated with:

IRS

On receive, record terminators (CR) are always returned for:

US CRC
IRS

irrespective of the setting of the mode flag.

6.7.6.5.6 **STX flag - (Memory location F859 or 63577)**

This flag is a single byte and controls the use of intermediate STXs. If the flag is set to non-zero, at the start of every intermediate record an STX is inserted. If the flag is zero, then STXs are not inserted except at the start of the first record in a block. Intermediate STXs are ignored on reception. The default of this flag is zero.

This feature is normally only used in 2780 installations.

6.7.6.5.7 **Retry flag - (Memory location F85A or 63578)**

This single byte flag controls how the protocol handles errors. If the flag is set to zero (its default state), then when an error in communication causes the HUNTER to re-try sending the last block, the HUNTER will re-try 10 times before giving a communication error. If, however, the flag is set to non-zero, the HUNTER will re-try indefinately until either it succeeds or the operator switches the HUNTER off.

6.7.6.5.8 **Space Compression flag - (Memory location F85B or 63579)**

This single byte flag controls the operation of space compression. If the byte is set non-zero, any string of more than 2 spaces is compressed into an IGS character and a space count code on transmission. The reverse happens on receive (i.e. the codes are expanded to the string of spaces). If the byte is set to zero (its default state) spaces are not compressed on transmission or expanded on receive.

This feature is not normally used in 2780 installations. It can be used to give increased throughput on a line where there are many spaces.

6.7.6.5.9 **Addresses of RAM Locations**

Name	Location (Hex)	Location (Decimal)	Default Value	Description
RECLEN	F853	63571	80	Record length
BUFLEN	F854	63572	200	Buffer length
SRFLG	F856	63574	0	Single record flag
RPFLG	F857	63575	0	Record padding
EMFLG	F858	63576	0	Mode flag
STXFLG	F859	63577	0	STX flag
RETRYS	F85A	63578	0	Retry flag
SCFLG	F85B	63579	0	Space compression flag

NOTE: These flags are preserved during power off and so will remain at their previous setting after subsequent power on.

FIG. 6.14 SYNCHRONOUS BUFFER ORGANISATION

Buffer Length

Record Length			Record Length			Record Length				
P A D	S Y N S	T X	DATA CHARACTERS			DATA CHARACTERS			E T B	C R C		

IRS or EM if selected

Record Terminator. Depending on the state of the mode flat, this would be either:
a) an IRS character, or
b) an IUS character followed by 2 intermediate characters.

An intermediate STX, if selected, would be inserted here.

(Note: If selected the STX would not be counted as part of the Record).

Line turnaround characters (FFH).

When calculating the buffer length, space should be included for the characters at the start of the buffer (i.e. PAD,SYNS and STX), the characters at the end (i.e. ETB or ETX, 2 CRC characters and 2 line turnaround characters) and all the required record terminators and intermediate STXs.

6.7.7 **USE IN TERMINAL EMULATION MODE**

Although 2780 work stations are not generally interactive devices, it is possible to use HUNTER in this mode when, for example, testing the system with LOG ON cards, etc.

When in Terminal Emulation any received data is displayed on the screen.

To transmit, a message character may be typed in directly. Provided no mistakes are made, and there is less than 5 seconds between each character, the message is sent on the transmission channel.

6.7.8 **COMMUNICATION ERROR**

There are a number of different line errors which may occur when using this protocol. These are detected and then displayed as communications errors. An error number is also displayed. The meaning of these error numbers is shown in section 6.7.5, MESSAGE TRANSMISSION. If use of the ON COMMS statement is made then the error number will be in the COMERR flag.

There is one exception to this rule, see Fig 6.2, UNANSWERED LINE BID. This is the most usual error and will generally be due to a connection fault, as no characters have been received. The option is then given to: Retry transmission ?(Y/N) if 'Y' is pressed then HUNTER will send out up to 10 more ENQs (enquiries) and the offer of retry made again. If 'N' is pressed then the error 01 is returned, and if programmed through ON COMMS, returns control back to Basic.

6.7.9 **ERROR MESSAGES:**

6.7.9.1 **Error 01: Unanswered line bid**

This is due to not receiving any reply in a request for the line, usually caused by incorrect connection of the communications channel.

6.7.9.2 **Error 02: No response from receiver**

After establishing the line, a buffer has been sent to which no response has been given, despite 10 ENQs having been sent.

6.7.9.3 **Error 03: Response not matched by odd-even block count**

The wrong positive acknowledgement has been given to a buffer, despite confirmation requests.

6.7.9.4 **Error 04: Retransmission rejected**

Repeated transmission of a buffer gets only the NAK response, has tried 10 times.

6.7.9.5 **Error 05: No ENQ after WACK**

Having sent out a WACK no acknowledging ENQ has been received for 5 seconds.

6.7.10 **HARDWARE HANDSHAKING**

This protocol is generally used with modems or modem simulators. The use of RTS (Request to Send) and CTS (Clear to Send) is therefore specified by the modem, particularly if half duplex or two wire operation is desired (as on the public switched telephone network). Both RTS and CTS may be deactivated as usual.

If they are used, then the following takes place:

1) If HUNTER is not transmitting then RTS is inactive, allowing the remote modem to transmit.

2) When starting a transmission buffer the RTS is activated. Before transmission starts CTS is awaited from the modem. When CTS becomes active HUNTER starts transmission.

3) During transmission RTS is kept active, but CTS is not checked, if it became inactive the transmission would be broken and synchronism dropped.

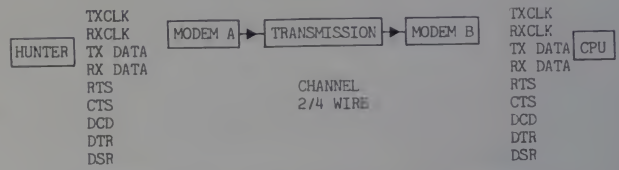
4) At the end of a transmission RTS is deactivated and HUNTER awaits reception.

There are no timeouts on the handshakes.

6.7.11 **HARDWARE CONFIGURATION**

The transmission channel could be the switched public network, 4 wire private lines etc.,

Fig 6.13 HUNTER TO MAINFRAME CONFIGURATION



The two modems and the channel could be a modem eliminator for nearby operation.

As can be seen the modems generally create all the clocking in this simple arrangement. HUNTER will not generate any actual clocking of its own, except when self synchronising to the data.

The handshake signals RTS and CTS ensure correct data flow over a half duplex line (allowing the modems to settle etc.) They may not be necessary over 4 wire links.

6.7.12 **HUNTER TO HUNTER CONNECTION**

To operate this protocol back-to-back between two HUNTER's it is only necessary to use a crossed 3 wire lead (TX,RX and ground). If the handshake lines are connected, then CTS must be deselected on the menu as the handshaking is only configured for modem operation.

HUNTER's need to be set for self clocking at the designed speed, typically 1200 baud or 300 baud over slow speed async modems.

TERMINAL EMULATION

6.8.1 **APPLICATION**

In terminal emulation mode, HUNTER can be used as:

a) A remote terminal for dialup timeshare computer services.

b) As a peripheral to other computers.

c) As a portable 'Telex', able to communicate with other HUNTER's or data terminals.

6.8.2 **CONFIGURATION**

HUNTER's RS-232 serial interface is configured to represent a Data Terminal Equipment (DTE) for compatibility with Modems and other items.

6.8.3 **OPERATION**

First, check that the communication port is correctly initialised, see section 6.4, COMMUNICATIONS PORT SOFTWARE.

6.8.4 **SELECTION**

From 'File Manager' select 'TERMINAL EMULATION' mode. You are now in HUNTER's remote computer terminal emulation mode.

HUNTER is now ready to communicate.

The function keys are programmed as follows:

Fn 4.....COMS
Fn 8.....SYST

Keyboard entry causes characters to be transmitted, including control characters see section 9.2, ASCII CHARACTER SET.

Characters sent to the HUNTER will appear on the screen if printable. Upper and lower case alphabets are available.

Exit from Terminal Emulation mode is made via the 'SYST' function key.

6.8.5 **PROTOCOLS**

Any protocols selected will be observed by HUNTER's communication software package automatically. Remember that block protocols like ACK/NAK will not transmit until 'ENTER' (CR) is pressed.

6.8.6 **Buffers**

It should be remembered that the HUNTER is fully buffered with transmit and receive communications buffers, a keyboard buffer and a virtual screen effectively keeping early information available for use.

At high receive data rates, e.g. 4800 baud, the receive buffer will remain full. If a "stop" is sent to the sending computer (e.g. XOFF or control S, sent manually) then the screen display will not stop immediately, continuing to display the characters stored in the receive buffer until the buffer empties. If this is found undesirable, the most straightforward solution is to reduce the data rate to 1200 baud.

If 2780 synchronous transmission is used in terminal mode, a block will be sent if the keyboard has not been operated for 5 seconds. This does mean that when using terminal emulation in 2780 mode for "signing-on", there should be no hesitation or typing errors or the block will be sent unexpectedly!

6.8.7 **Auto Power Off**

Terminal Emulation does not power off automatically. This is for ease-of-use when operating intermittently with a remote site. If auto power off were allowed, modems may drop the line resulting in messages lost, etc.

It is important, therefore, to remember to switch off HUNTER after using Terminal Emulation.

6.8.8 **System Call**

User programs can use Terminal Emulation which is provided as system call number 43 (see section 3.5.3.44). Selecting "System" (function 8) in Terminal Emulation will return control to the calling program.

To use Terminal Emulation from Basic, the following is required:

```
1000  A=ARG(43):A=CALL(5)
```

Execution will continue after line 1000.

From a machine code or compiled program, a system call 43 is required, e.g:

```
.
.
LD   C,43   ; select call
CALL 5
.
.
.
```


6.9 COMMUNICATIONS ERRORS

HUNTER provides a range of communications error messages to warn the user of failure situations. The error messages are designed to be trapped by the Basic ONCOMMS statement (see section 5.16.3) and handled by the user's program.

However, the errors can occur when HUNTER is under manual control (for instance, in TERM mode) or if the user's application does not contain ONCOMMS. In this event, HUNTER will display:

* * * Communication Failure * * *

\- - Error nn - -

where nn is the Error number. The attempt at communication can be aborted either by powering the HUNTER off or by pressing ENTER, in which case control is passed back to the DEMOS file manager.

The available error messages are:

6.9.1 Error 01: Unanswered line bid

This is due to not receiving any reply in a request for the line, usually caused by incorrect connection of the communications channel. (Synchronous mode only).

6.9.2 Error 02: No response from receiver

After establishing the line, a buffer has been sent to which no response has been given, despite 10 ENQs having been sent. (Synchronous mode only).

6.9.3 Error 03: Response not matched by odd-even block count

The wrong positive acknowledgement has been given to a buffer, despite confirmation requests. (Synchronous mode only).

6.9.4 **Error 04: Retransmission rejected**

Repeated transmission of a buffer gets only the NAK response, has tried 10 times. (Synchronous mode only).

6.9.5 **Error 05:** Loss of response from receiver.

No ENQ has been received in reply to a WACK (synchronous mode only).

6.9.6 **Error 06:** Not used.

6.9.7 **Error 07:** Not used.

6.9.8 **Error 08: Receive timeout Error**

No data has been received to a request for input (e.g. LINPUT) for the specified time.

6.9.9 **Error 09: Transmit timeout Error**

Transmission has not been able to take place for the specified time.

The error number is available in the location COMERR (see section 9.7, page 9-38).

6.9.10 **Error 10: ETX/ACK timeout Error**

No response has been received for 30 seconds.

6.9.11 **Error 11: ACK/NAK failure**

Three attempts have failed to get an ACK.

6.9.12 **Error 12: Systime failure**

Ten re-transmissions have failed to get an ACK.

EDITOR

CONTENTS

INTRODUCTION

7.1 Hunter's built-in text editor provides practical text handling facilities.

It may be used for the creation and alteration of standard ASCII text files for reports, correspondence, etc. This feature is invaluable for text entry 'on the move' in trains, planes or any other form of transport. Text may then be easily printed out on return to the office.

Husky Basic files, which are encoded to save memory and execution time, may also be edited directly. These files are recognised by their use of the .HBA suffix. It is important, therefore, to maintain the .HBA suffix convention when attempting to edit Basic programs.

NOTE: Encoded source files for other interpreted programs e.g. MBASIC cannot be edited with this editor.

7.1.1 **Automatic Cont Edit**

On entry to the Editor from either Basic or Demos, the HUNTER is forced into Cont mode. This applies even if the Editor has been initiated but the HUNTER is still converting a Basic tokenised program into its text equivalent and the normal Editor screen is not yet displayed.

If the HUNTER is now powered down, either by manually pressing the power key or by auto-timeout due to lack of keyboard operation, then on power up the HUNTER re-enters the Editor and continues operation from where it ceased at power down. This applies in all cases e.g. midway through a search, writing to a file or exiting to Basic and recreating a Basic tokenised file from its text equivalent. This feature ensures that all data entered into the Hunter whilst in the Editor is always accessible to the user. On exit from the Editor, either by 'Exit' or 'Save', the HUNTER is forced out of Cont mode automatically. Thus, if the HUNTER is powered down, then it will power up normally.

7.1.2 **Editor program size check**

When the Editor is initiated, an automatic check on the file size is carried out. Since the Editor uses only the TPA for its work space, if a text file greater in length than the TPA is attempted to be edited, a warning message will be displayed and the operation will be aborted.

NOTE: The file will not have been changed in any way and the Hunter will not be in auto Cont mode.

In the case of editing a Basic tokenised program, a check is carried out on the size of the tokenised program so that its text equivalent form would not be greater in length than the size of the TPA.

Basic programs are tokenised so as to reduce the amount of memory required to store the program and increase the speed of operation of the Basic program. On converting a Basic tokenised program into its text equivalent, the size of the program increases by approximately 1/3. Hence, although the size of the Basic tokenised program (obtained by using the STAT command in DEMOS) indicates that it would fit into the TPA, this, of course, does not imply that its text equivalent would fit into the TPA and, therefore, it cannot be assumed the Basic tokenised program can be edited.

The maximum size of a Basic tokenised program which can be edited is approximately 36K bytes. Hence, for normal applications this size check does not impose limitations upon the HUNTER.

This page intentionally left blank.

INITIATING THE EDITOR

The editor can be invoked from the operating system, DEMOS, or directly from BASIC. In either case, procedures are similar but, on exit, the user will be returned either to the operating system or Basic as appropriate.

7.2.1 **Entry from the operating system, DEMOS**

From DEMOS, either type EDIT, followed by the desired filename, or use function key 7 followed by the filename.
The entry line will look like:

```
>EDIT myfile.txt
```

This line will edit the file myfile.txt. Pressing Enter will invoke the Editor and if the file already exists, will load it into the work area. If the file does not exist, it will be created on exit from the editor provided text entry has been made.

It is essential to enter a filename. If the suffix is omitted, the file will be assumed to be ASCII and have the standard default extention TXT. If it is desired to edit a Basic file, the .HBA suffix must be entered.

Assuming myfile.txt does not exist, the screen will show:

```
+----------------------------------------+
|COL:00    INS:on     Caps:off           |
|                                        |
|                                        |
|                                        |
|                                        |
|Caps Find Cins                Save Ldel Exit|
+----------------------------------------+
```

On the top line there are three items of information:

a) **COL:00**

This shows the column which the cursor is in. It is in the range 0 to 79. As the cursor moves, it is kept updated and may be used for aligning tables, indents in text, etc.

Column 0 is the left of the virtual screen. Column 79 is the extreme right of the virtual screen.

b) **INS:on**

This shows whether character insert is on or off.

If insert is on, any text entered will be placed at the cursor position and any text already present will be moved. With insert off, the display will show:

```
INS:off
```

In this case, entries will overwrite existing text.

c) **Caps:off**

This shows the state of the CAPS lock function which operates as described in section 4.5.

NOTE: The CAPS LOCK inverts upper and lower case alphabetic characters, but is not a 'Shift Lock' function. Lower case characters are available by pressing 'Shift'.

Free-format text entry may now proceed as required. The cursor keys provide up, down, forward and backspace functions. Lines are terminated by 'Enter' which is equivalent to CR (Carriage Return).

If the file already exists, the first few lines of text will appear on the screen.

```
+----------------------------------------+
|COL:00 INS:off Caps:off                 |
| This is a demonstration of entering tex|
|the text editor. On this screen it can b|
|we only display the top left of the file|
|                                        |
| Paragraphs are generally indented, this|
|no problem to the Hunter text Editor.   |
|Caps Find Cins       Save Ldel Exit     |
+----------------------------------------+
```

This is the beginning of the file and the left-hand forty columns. Text editing may now proceed as required.

NOTE: Exit from the editor after entering from DEMOS will abort any changes to the file, unless the file is previously saved.

7.2.2 Entry from BASIC

When in Basic (see section 4), it is possible to use the Editor to load a file or to Edit a program which has already been loaded into the TPA.

To Edit a Basic program already resident in the TPA, it is only necessary to type:

EDIT (followed by 'Enter')

or

Fn key 3 (followed by 'Enter')

There will be a slight delay while the Basic program is un-entokened to give an ASCII representation. The screen will then appear as shown above and editing can proceed as required.

Alternatively, if a new file is to be loaded into the workspace, entering:

Edit "filename" (followed by 'Enter')

If the file extension is omitted, the .HBA extension will be assumed and will first load the file and then invoke the editor.

When exiting from the Editor after entering from Basic, use of EXIT will cause the ASCII character representation used by the Editor to be entokened ready for execution by Basic. This procedure can take a few seconds for larger programs.

After EXIT, the user is returned to Basic.

NOTE: Any unnumbered lines will not be entokened and consequently will be lost. Any duplicated line number will cause the earlier entry to be lost.

EDITOR OPERATION

7.3.1 The Editor is incorporated in the operating system as a built-in function and operates as a file-based system.

When a file is loaded into the Editor, it is placed in the TPA (see section 3.3), the 54K workspace used by HUNTER. This is the only area in which the Editor operates on a file. The whole file must be loaded.

The maximum text file size wich can be edited is 54K bytes.

7.3.2 Basic Files

If an encoded Basic file is loaded, then it is first un-entokened to ASCII characters. This is because the Editor will only operate in ASCII within its work area. A short delay of a few seconds on entry and exit to the Editor will occur when operating on Basic files. This delay obviously gets longer with larger programs.

7.3.3 The Keyboard

The diagram of the keyboard shown in Fig 7.1, shows the function associated with each key when used as a control key (i.e. hold down CTL/FN and press the appropriate key).

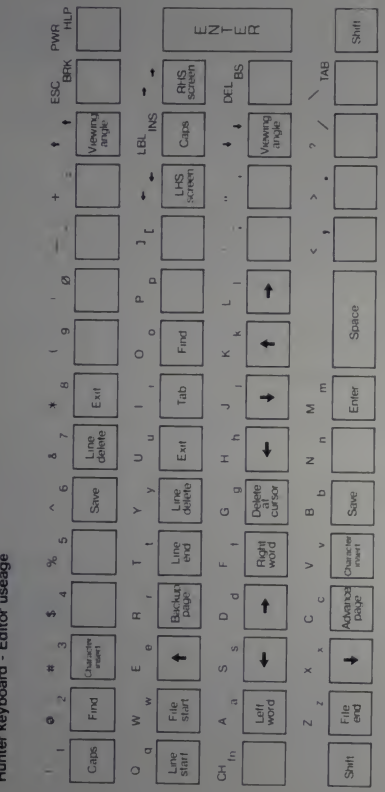
All the function keys are replicated as control keys which conform broadly to the layout used by Wordstar™.

NOTE: The Editor uses the virtual screen, so the cursor keys (fully described in section 2.6) may be used to scan around the text in the virtual screen.

The label line cannot be turned off with LBL when in the Editor.

The operating system scans the keyboard every 18ms and stores the key code in a 32-key keyboard buffer. This buffer is examined by the user's program and operating system whenever necessary. When in the Editor mode, the operating system is examining this buffer and taking appropriate action when necessary. There are occasions when the screen handler cannot keep pace with keyboard entries due to considerable and repetitive editor workspace access. This may become a problem when an excessive number of character delete key operations are performed. It is, therefore, advisable not to invoke the auto-repeat function when only a limited number of deletes are required.

FIG. 7.1



7.3.4 Line Length

The Editor supports lines of greater than 80 characters by wrapping round onto the next line.

EDITOR COMMANDS

7.4.1 Commands made through the function and control keys may generally be made at any time.

Each group of commands will be described individually. Reference should be made to Fig 7.1 which shows the complete keyboard functions available.

NOTE: Control and function keys are denoted with a leading up 'carot' (^), e.g. Control G is shown as ^G and function 2 as ^2.

ARROWS

7.4.2 **KEYS:**

↑;^E;^K	-	Up arrow
↓;^X;^J	-	Down arrow
←;^S;^H	-	Left arrow
→;^D;^L	-	Right arrow

FUNCTION: These keys move the cursor non-destructively in the required direction. They may be used to place the cursor over any character position which is in use.

These keys will not place the cursor outside the text area, i.e. before the beginning of the file, after the end of the file, or past the end of the line currently written to.

REMARKS: The left and right arrow keys will scroll to the previous/following line if the end of a line is reached.

When 'shifted', these keys access the virtual screen.

When pressed with "CTL/FN", they have the following functions:

1) Up and down cursor keys adjust display contrast.

2) Left and Right cursor keys display left and right halves of the virtual screen.

CAPS LOCK & TAB

7.4.3 **KEYS:**

^LBL	-	Caps Lock
^1	-	Caps Lock
TAB	-	Tab
^I	-	Tab

FUNCTION: CAPS lock forces all alphabetic characters to capitals. This function is shown on the top line of the display. Pressing caps lock a second time turns the function off.

TAB moves the cursor onto the next tab position in units of 8, i.e. column 8, 16, 24, etc.

REMARKS: CAPS lock is useful for entering programs.

CHARACTER INSERT

7.4.4 **KEYS :**

^3	-	Character Insert
^V	-	Character Insert

FUNCTION: This key enables and disables character insert. The condition of the function is shown on the top line, e.g.

INS:on

When turned off, characters entered overtype existing text.

When turned on, new text is inserted before the cursor position. This is useful for adding characters to the middle of a line.

REMARKS: An exception to the above rule is carriage return (Enter). With insert on, a carriage return is added as with any other character, causing a break of line if the cursor is in the middle of the text. With insert off, the carriage return is only added to the file if it is at the end of the file, otherwise the cursor is moved non-destructively to the beginning of the next line.

DELETE

7.4.5 **KEYS:**

DEL	-	Delete
BS	-	Back space
^G	-	Delete ahead

FUNCTION Delete destructively moves the cursor back over the previous character. The character to the left of the cursor is removed and the rest of the line is moved one character to the left.

Backspace operates exactly as back arrow. It moves the cursor non-destructively one character to the left.

Delete ahead destroys the character on which the cursor is positioned. The cursor remains stationary and text ahead of the cursor is moved towards it.

REMARKS: Carriage return may be deleted. This has the effect of concatenating lines.

WARNING: Because HUNTER has a 32 character keyboard input buffer, 'Deletes' can be stored up in auto-repeat mode more quickly than they appear on the screen. This will have the effect of 'eating' the text!

Do not hold the Delete key down. Instead, use single key depressions or Line Delete (see section 7.4.9).

EXIT

7.4.6 **KEYS:**

^8	-	Exit
^U	-	Exit

FUNCTION: Exit operates slightly differently, depending on whether the editor was entered from either DEMOS or Basic. In either case, pressing Exit clears the label line and presents the question:

Exit Y/N?

answering N aborts the command and returns the cursor to its previous position.

Answering Y when entered from DEMOS exits from the editor with no change to any file. Any alterations are lost.

Answering Y when entered from BASIC re-processes the file into BASIC encoded form and leaves the edited Basic source in the workspace for execution.

When Exit is in progress:

** Wait

is displayed on the bottom line.

FILE START AND END

7.4.7 **KEYS:** ^W - File start
^Z - File end

FUNCTION: These commands place the cursor non-destructively at the beginning or end of the file.

File start : positions the cursor on the very first character, as if the Fditor had just been entered.

File end : places the cursor at the end of the file. The screen displays the last line of the file and puts the cursor on the second text line of the screen.

REMARKS: Very useful for adding text to a file.

FIND

7.4.8 **KEYS:** ^2 - Find
^0 - Find

FUNCTION: Find is used to seach the file for a specified string. The search is only forward from the cursor position to the end of the file. There is no search prior to the cursor.

After activating Find, the label line is cleared and the message:

Find?

is displayed. The desired string is typed in (up to 68 characters). After pressing Enter, the search is started and the message:

** Wait

is displayed. If an exact match for the desired string is found then the cursor is placed on the first character.

If no match is found, the message:

** Failed

is displayed on the bottom line and the cursor returned to its original position.

If it is desired to find the same string a second time, then pressing only Enter after the "Find?" prompt will cause the search to go on through the file looking for the next occurrence. This should not be done for the first search after entering the Editor as the command will be rejected.

REMARKS: Find can be aborted by entering ESC.

The "failed" message disappears after any key depression.

LINE DELETE

7.4.9 **KEYS:** ^7 - Line Delete
^Y - Line Delete

FUNCTION: Line delete removes the line on which the cursor is placed.

The cursor must be placed at the beginning of the line to be deleted.

LINE START AND END

7.4.10 **KEYS:** ^Q - Line Start
^T - Line End

FUNCTION: These keys move the cursor non-destructively to either the beginning or end of the line.

REMARKS: Useful for adding text to the end of a line.

PAGE SCROLL

7.4.11 KEYS: ^R - Page Up
^C - Page Down

FUNCTION These keys are used to scroll the text page by page.

Page down will firstly place the cursor at the bottom of the screen being displayed. Further operations will display pages six lines at a time to the end of the file.

Similarly, page up will initially place the cursor on the top line of the display and then backwards page by page to the beginning of the file.

REMARKS: Useful for reviewing text.

SAVE

7.4.12 KEYS: ^6 - save
^B - save

FUNCTION: This command is for saving modified text into a file.

After entering the command, the label line is cleared and the message:

Save Filename ...

is displayed. If 'Enter' is pressed immediately, the text is stored back into the file originally loaded. Otherwise, a new file is created with the text. However, if the Editor was entered from Basic by simply pressing Edit and then Enter, then a new file called 'HUNTER.HBA' will be created unless a filename is specified.

If .HBA suffix is used, then the file will be assumed to be a Basic source file. Any other suffix will always create a text file.

REMARKS: If there is insufficient room, then DEMOS is re-entered and the error message:

Disk Full File length = n blocks

is displayed where n is the number of blocks needed to save all the file currently in the Editor. By clearing sufficient space in the HUNTER it is possible to save the file in the Editor using the file manager command:

SAVE n filename

where n is the same number displayed in the error message.

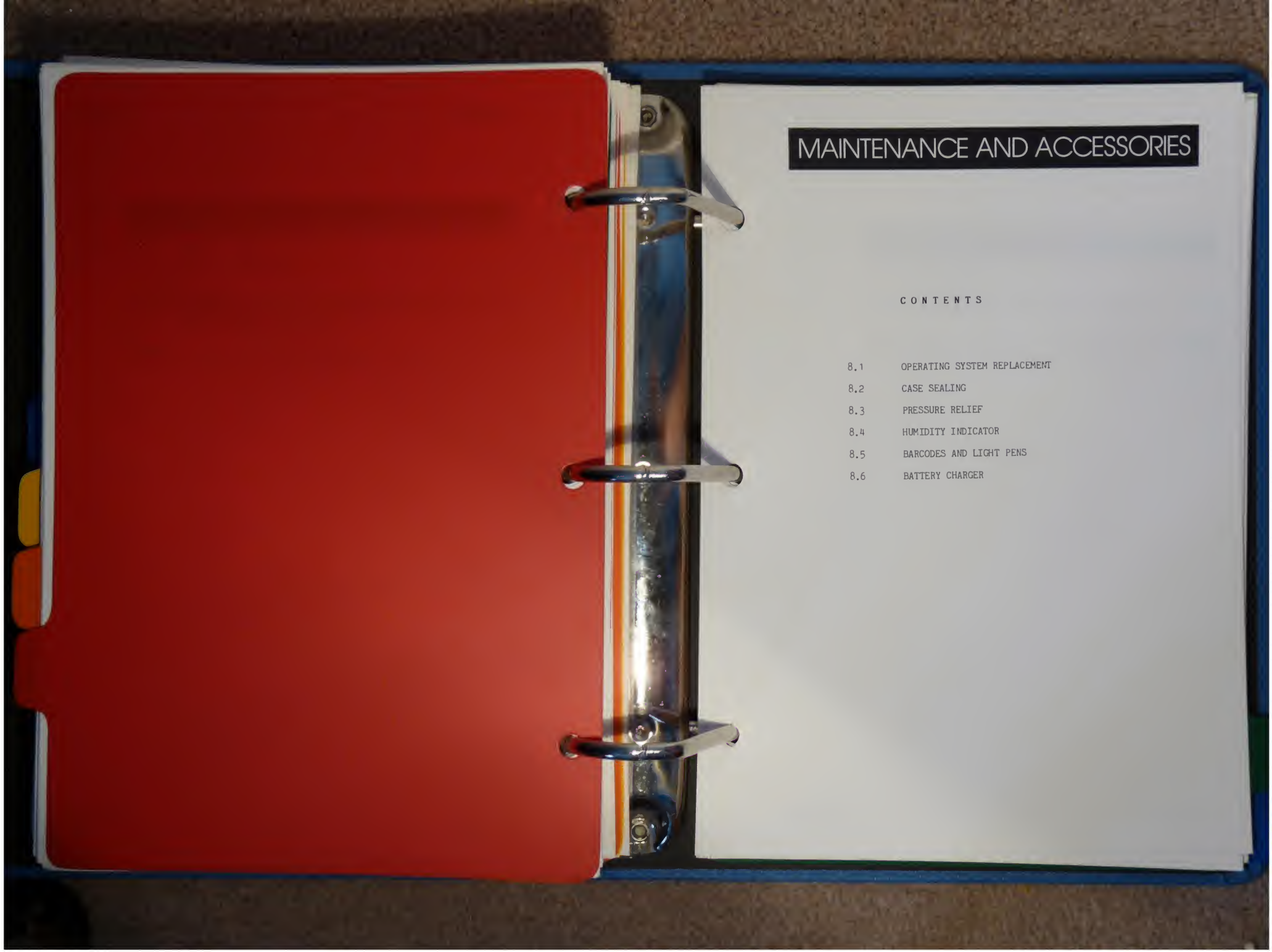
This will happen even if the Editor was entered from Basic.

WORD SKIP

7.4.13 **KEYS:** ^A - Left word
^F - Right word

FUNCTION: These keys move the cursor non-destructively left and right pausing only on spaces. Otherwise, they operate exactly as left and right arrow keys.

REMARKS: A fast method for gaining position in text.


MAINTENANCE AND ACCESSORIES
CONTENTS
8.1 OPERATING SYSTEM REPLACEMENT
8.2 CASE SEALING
8.3 PRESSURE RELIEF
8.4 HUMIDITY INDICATOR
8.5 BARCODES AND LIGHT PENS
8.6 BATTERY CHARGER

OPERATING SYSTEM REPLACEMENT

8.1

HUSKY Computers pursue a policy of continuous improvement in HUNTER's performance.

Users may wish to update their version of HUNTER to a later enhancement by replacing the operating system ROMs.

The unit should be returned to HUSKY Computers or authorised agents, where the enhancements will be made by experienced engineers and returned in the shortest possible time.

CASE SEALING

8.2.1 **STANDARD HUNTER**

All HUNTER's are built with integral seals to provide a high degree of protection against atmospheric moisture, corrosive gases and accidental immersion, therefore HUNTER should not be opened.

8.2.2 **THE BATTERY SEAL**

This is a small 'O' ring seal running in a machined groove in the battery plug and seating on a machined face in HUNTER casting.

This seal requires a smear of silicon grease to assist in seating.

8.2.3 **SEAL SPECIFICATION**

Dimension	Part no.
1.5mm X 30mm	HUN-05710

Replacement seals may be obtained from HUSKY Computers or authorised agents.

PRESSURE RELIEF

8.3 Under certain circumstances (change of altitude, etc.) the interior of HUNTER may have a substantial pressure difference compared with the environment. This can lead to two consequences in HUNTER.

1) Internal overpressure : making the keys stiff and difficult to operate.

2) External overpressure : a tendency to ingest any rainwater adjacent to the seals, and in extreme cases, the possibility of activating the keys.

In either case, any pressure differential can be released by slightly loosening the battery cap.

If you are taking HUNTER on an airline trip, this is a sensible precaution to take.

HUMIDITY INDICATOR

8.4 HUNTER's interior is kept dry by an internal store of silica gel desiccant, kept in a muslin bag. The dryness of HUNTER's interior is essential to its correct operation, and is indicated by a humidity indicator in the corner of the display window.

The colour of the indicator should be checked at intervals.

THE HUMIDITY INDICATOR MUST ALWAYS BE BLUE

If it takes on a pinkish colour, the unit must be returned to HUSKY Computers promptly to avoid corrosion of internal parts. If the colour changes immediately after an accident involving water or high humidity, a fault has occurred in the sealing, and there could be water inside the unit.

If you suspect that water has entered the interior, you **MUST** take the following action:

1) Remove the battery

2) Return HUNTER to HUSKY Computers, or authorised agents.

BAR CODES AND LIGHT PENS

8.5.1 As an option HUNTER is capable of reading 2 types of barcodes:

a) Code 39

b) EAN 8/13

To provide these facilities software packages are loaded into the file system of HUNTER as permanent files. Access to these files is not allowed to the user. However, the operating system accesses the files in order to support the wand option when required in programs utilising the Basic statements such as WINPUT, WINCHR etc. Access to the files is totally transparent to the user.

Other codes are available to special order.

If an attempt is made to access these files without them being present, then a "system file error" will be displayed.

See section 9.9.3, LEMO CONNECTOR, for details of connecting and removing the LEMO plug.

8.5.2 **BAR CODE SCANNING TECHNIQUES**

There are a few simple rules to follow when using a hand held wand:

Check that the tip of the wand is free from dirt. Prolonged use of the wand may lead to a build up of dust inside the tip, covering the lens. To check for this the tip must be unscrewed and, if dust is found, a gentle wipe with a soft cloth will remove it. The performance of the wand will not be immediately affected by a build up of dust, but rather a gradual decrease in the reliability of the wand operation. A weekly check should be all that is required for normal use.

The tips of certain wands are made of plastic and as such, pressing the wand firmly onto the bar code will result in tip wear. This will affect the wand performance since the focal length of the lens will not coincide with the distance between the barcode and the lens, i.e. the bar code will appear to be 'out of focus'. It is recommended that the wand be held gently in the hand and moved lightly across the bar code for best results.

It is recommended that the scan should be carried out at a constant speed with the wand in contact with the barcode

throughout the scanning period. The scan should be at right angles to the bars of the bar code. However, HUNTER tolerates a considerable variation in the scan technique.

The scan can be slow, fast, traversed across the bar code in an arc rather than a perfect straight line, or even scanned in a 'wavy' line, provided that the wand remains within the bar code region. HUNTER will allow both Code 39 and EAN 8/13 bar code types to be scanned either left to right or right to left.

NOTE 'Code 39' is the trademark of Intermec Corp.

It is not necessary to place the wand precisely at the beginning of the bar code before a scan since HUNTER will work out which information transmitted from the wand actually corresponds to the bar code itself. If the wand is resting on a bar code, it is possible to move the wand to either end of the bar code and then scan in the opposite direction. HUNTER will then work out the beginning and end of the bar code.

HUNTER will allow a maximum of 16 characters to be represented by a Code 39 bar code. This should be borne in mind if the reader intends to produce his own Code 39 bar codes. If larger codes are required, contact HUSKY Computers for details.

8.5.3 EUROPEAN ARTICLE NUMBER, EAN 8/13

There are 2 EAN bar code formats:

a) EAN 8 Short Version

b) EAN 13 Full length Version

Both a) and b) are used in marking retail articles of sale in shops, Hypermarkets, warehouses, etc.

EAN 13 is the general name used to describe a series of barcode formats of which ANA (United Kingdom) is one particular version. See TABLE 8.1, ASSIGNMENT OF PREFIX DIGITS BY EAN, for precise details.

The general form of EAN 13 is 13 all-numeric digits comprising:

First 2 digits

Prefix denoting the National Numbering Authority administering the remainder of the number.

Next 10 digits

National Article Number, the structure of which is determined by the National Numbering Authority.

Last digit

Check digit, calculated by modulo-10 arithmetic, i.e:

Prefix	National Article No.	Check
P1 P2	xxxxxxxxxxxxxxxxx	C
eg:	5000183962862	

In the above example the prefix is 50, i.e. the numbering authority is ANA, the United Kingdom's authority. The National Article Number is 0018396286 and the check digit is 2.

The EAN 8 system is an entirely independent series of numbers of 8 digit length. The general form of which is:

First 2 digits

Prefix, as in EAN 13

Next 5 digits

National Short Article Number

Last digit

Check digit, as in EAN 13

Prefix	National Article No.	Check
P1 P2	xxxxxxxxxxxxxxxxx	C

e.g:

50159109

where, the prefix is 50 and so the National Numbering Authority is ANA. The Short Article Number is 15910, and the Check Digit 9.

The general format of EAN 13 is shown in Fig 8.2, 12 CHARACTER BAR CODE. As can be seen, there are 3 types of guard patterns; 2 normal guard patterns and a centre guard pattern. Only 12 of the 13 digits are represented by barcodes and the 13th digit must be calculated by considering the mixture of characters which represent the first 6 digits. The first 6 digits are represented by characters chosen from either Set A or Set B, whilst the last 6 digits are represented by characters from Set C only. See Fig 8.3, CODING OF NUMBER CHARACTERS. The allowed combinations of Set A and Set B characters are shown in TABLE 8.4, COMBINATION OF SET A AND SET B CHARACTERS.

The general format of EAN 8 is shown in Fig 2.3, 8 CHARACTER BAR CODES. Note in this case that the first 4 digits are all chosen from Set A and the last 4 digits from Set C. Again, the guard bars are present.

Due to the fixed parity (all from Set C) of the last 6 digits in EAN-13 and the last 4 digits in EAN-8, both types of barcodes may be scanned in either direction, i.e. the barcodes are bi-directional.

TABLE 8.1 ASSIGNMENT OF PREFIX DIGITS BY EAN

Prefix Values	
00-09	(Reserved for UPC)
20-29	In-Store Numbers
30-37	Gencod (France)
40-43	CGS (Germany)
49	Distribution Code Centre (Japan)
50	ANA (United Kingdom)
54	ICODIF (Belgium)
57	Dansk Varekode Administration (Denmark)
61-62	(Reserved for DCI)
64	The Central Chamber of Commerce (Finland)
65-69	(Reserved for DCI)
70	(Norway)
73	Swedish EAN Committee (Sweden)
76	Schweizerische Artilelkode Vereinigung (Switzerland)
77	APNA Australia
80-83	(Italy)
84	AECOC (Spain)
87	UAC (Netherlands)
90-91	BAN -Austria
978	ISBN
979	Reserved for ISBN
98-99	Coupon Numbers

Fig 8.2 12 CHARACTER BAR CODE

Fig 8.3 CODING OF NUMBER CHARACTERS

VALUE OF CHARACTER	NUMBER SET A (odd)	NUMBER SET B (even)	NUMBER SET C (even)
0			
1			
2			
3			
4			
5			
6			
7			
8			
9			

TABLE 8.4 COMBINATIONS OF SET A. AND SET B. CHARACTERS.

Value of 13th Digit	Number Sets used for Coding left half of symbol					
	1	2	3	4	5	6
0	A	A	A	A	A	A
1	A	A	B	A	B	B
2	A	A	B	B	A	A
3	A	A	B	B	B	A
4	A	B	A	A	B	B
5	A	B	B	A	A	B
6	A	B	B	B	A	A
7	A	B	A	B	B	A
8	A	B	A	B	B	A
9	A	B	B	A	B	A

Fig 8.5 8 CHARACTER BAR CODE

8.5.4 **CODE 39**

CODE 39 is an alphanumeric bar code consisting of 43 data characters (0-9, A-Z, 6 symbols and space) and a unique start/stop character "*". These characters are represented by light and dark bands, as shown in Fig 8.5, CHARACTER BAR CODE.

Code 39 is so named due to the structure of each character being represented by 9 elements (5 bars and 4 spaces) 3 of which are wide and the remaining 6 are narrow. A wide bar or space is assigned a value 1, and a narrow bar or space is assigned a value 0. Gaps between characters have no value.

The number of characters in a code is limited only by the capabilities of the reader equipment or by human factors if a hand held wand is used.

Each code starts and ends with an asterisk, "*", thus the code may be scanned in either direction.

The width of a unit bar may vary considerably allowing various printing methods to be used to produce the characters, e.g: offset, letter press, dot matrix printers. The recommended standard density is 9.4 characters per inch, but a density of 1.4 characters per inch may be used for corrugated containers.

A complete Code 39 bar code consists of a leading white space (referred to as a quiet zone), a start character, data characters, a stop character and a trailing quiet zone.

A check digit may be produced if required and is modulus 43. The check digit is the last data character and calculated in the following manner:

Suppose the data characters are:

12345ABCDE/

then the sum of the data characters are:

1+2+3+4+5+10+11+12+13+14+40=115

Now, 115/43=2 Remainder 29

The check digit is the character corresponding to the value of the remainder, which in this example is 29, i.e. "T".

Therefore, the complete data character sequence is:

12345ABCDE/T

The numeric values assigned to each Code 39 character is shown in TABLE 8.6, CODE 39 CHARACTER VALUES.

Fig 8.6 CODE 39 CONFIGURATION

CHAR	PATTERN	BARS	SPACES	CHAR	PATTERN	BARS	SPACES
1				M			
2				N			
3				O			
4				P			
5				Q			
6				R			
7				S			
8				T			
9				U			
A				V			
B				W			
C				X			
D				Y			
E				Z			
F				-			
G				.			
H				SPACE			
I				*			
J				$			
K				/			
L				+			
				%			

The * symbol denotes a unique start/stop character which must be the first and last character of every bar code symbol. Note that the start/stop character is distinct from the "asterisk" defined in Table 8.

TABLE 8.7 CODE 39 CHARACTER VALUES

CHAR.	CODE	CHAR.	CODE	CHAR.	CODE
0	0	F	15	U	30
1	1	G	16	V	31
2	2	H	17	W	32
3	3	I	18	X	33
4	4	J	19	Y	34
5	5	K	20	Z	35
6	6	L	21	-	36
7	7	M	22	.	37
8	8	N	23	Space	38
9	9	O	24	$	39
A	10	P	25	/	40
B	11	Q	26	+	41
C	12	R	27	%	42
D	13	S	28		
E	14	T	29		

BATTERY CHARGER

THESE INSTRUCTIONS RELATE TO PERSONNEL SAFETY AS WELL AS RELIABLE OPERATION OF HUNTER AND BATTERY CHARGER. IT IS IMPORTANT THAT THEY ARE READ AND UNDERSTOOD.

8.6.1 **DESCRIPTION**

HUNTER's rechargable battery system consists of four rechargable cells and a HUSKY HUNTER battery charger.

The cells are 'AA' size Nickel Cadmium (NiCad) of nominally 1.2 volts, 0.50 Ah and are capable of being charged at 65mA continually.

The special HUSKY HUNTER battery charger is the only unit which may be used to recharge the NiCad batteries when they are installed in HUNTER. The use of an ordinary AC adaptor/battery eliminator may damage HUNTER or the batteries.

8.6.2 **W A R N I N G**

When using the charger be absolutely certain that HUNTER has four NiCad cells correctly aligned and no alternative type is present. Do not even mix NiCads of differing age or state of charge. A risk of chemical leakage, gassing or explosion exists if anything other than a matched set of recommended NiCad cells are charged. Do not attempt charging at temperatures of 5°C or less.

The charger contains lethal voltages, under no circumstances must it be opened, in any way tampered with, or used for any other purpose. It must, of course, be kept dry. There are no user serviceable items inside the case.

If the charger is suspected of being faulty, then the fuse inside the mains plug may be changed for a similar fuse not exceeding 2 amps rating. The charger unit is fully protected against continuous short circuits but if unlikely fault conditions arise which cause overheating to the charger, it will self destruct, quietly and safely. The charger is double insulated, making no earth connection.

8.6.3 **OPERATION**

To use the charger unit the following procedure is recommended:

1) Switch off HUNTER.
2) Plug the charger into HUNTER's LEMO connector.
3) Plug the charger into the mains.
4) Finally switch mains on.

See section 9.9.3, LEMO CONNECTOR, for details of connecting and removing the LEMO plug.

HUNTER operation is possible during use of the charger but the current drain will counteract the charging current and consequently prolong the time to fully charge the batteries. Removal of the cells under this condition is forbidden.

As with all electrical apparatus the charger should be disconnected from the mains when not in use. Only 240 vAC (220 to 250v) mains, 50 or 60 Hz is suitable for standard units. Chargers for alternative mains supplies, e.g. 110v, are available.

Exhausted batteries will typically be fully charged in 12 hours, overnight charging is a popular practice.

8.6.4 **NICAD REPLACEMENT**

It must not be forgotten that rechargable batteries do not have an unlimited life and will ultimately need replacing as do car batteries. If the batteries become exhausted after little use then the set may need replacing. Their life expectancy can be several years but this is reduced by repeated or prolonged total discharge and by excessively high temperatures.

NOTE: Several discharge/charge cycles are required before NiCad cells reach their peak capacity. This type of battery will also self discharge, especially at elevated temperatures, which can result in a fully charged set of batteries becoming flat in a few weeks. An occasional charging session, say, every fortnight is recommended if HUNTER is being stored.

8.6.5 **MAINS OPERATION**

For critical or remote operations, permanent charging may be called for. This is acceptable provided ventilation around the charger and HUNTER is good, but ageing of the cells may be accelerated. Annual replacement would be a prudent action in this instance.

APPENDIX

CONTENTS

HUNTER SPECIFICATION

9.1.1 **PHYSICAL**

Construction	:Diecast aluminium alloy.
Size	:216mm x 156mm x 32mm.
Weight	:(Including batteries) 1150 grammes.
Sealing	:Waterproof against accidental immersion.
Straps	:Wrist strap on either side.

9.1.2 **FACIA**

Screen	:240 x 64 dot full graphic liquid crystal display with keyboard controlled contrast adjustment.
Keyboard	:58 keys arranged in four rows of 15 Qwerty layout with function and control keys. Fully waterproof sealed rubber keys.

9.1.3 **PROGRAMMING**

Built-in language	:Extended Basic interpreter.
File handler	:CP/M compatible software environment allows loading of many popular programs.
Storage	:Programs and data stored indefinately in battery supported memory. Operating system stored in firmware, occupying no RAM space.
Screen handling (text mode)	:The actual screen of eight 40 character lines forms a window on a larger virtual screen. The window is controlled with the cursor keys.

	Screen handling (graphics mode)	:Both text and graphics information are displayed together. There are five selectable character sizes giving facilities for elegant, professional screen formatting.
	Keyboard	:All keys re-allocatable in software.

9.1.4 **COMMUNICATIONS**

Type	:RS-232/V24 serial port on standard 25 pin 'D' type connector.
Configuration	:Entirely software controlled.
Baud rate	:Up to 4800 baud asynchronous.
Protocols	:Standard 'invisible' protocols for flow control and security are transparent to user programs. Formats provided include: (a) none/simple 'TTY' communication, (b) XON/XOFF for mini and main frame computer systems, (c) ETX/ACK for many popular printers, (d) ACK/NACK for secure telephone communication, (e) Systime, with BCC error checking, (f) 2780 fully synchronous implementation.
Handshaking	:RTS, CTS, DTR, DSR, RI, CD, all selectable by software.

9.1.5 **GRAPHICS**

Full addressing of all 240 x 64 dots	:Software support for lines, boxes, circles and points.

9.1.6 **SOUND**

Fully programmable for 4 octave range note and duration.

9.1.7 **CHARACTER SETS**

Five different sized charactersets selectable from Basic. Reverse video characters supported for all sets.	:True descenders on larger sizes.

9.1.8. **MEMORY**

Type	:CMOS low power semiconductor RAM.
Retention	:Battery support with soldered in Ni/Cd backup.
Capacity	:Available in RAM size options of 80k byte, 144k byte and 208k byte. Operating system utilises 8k byte of this memory.
Firmware	:Built-in operating system including file manager, Basic interpreter and disk emulator.
Upgrading	:RAM memory can be upgraded on return to factory to larger size

9.1.9 **REAL TIME CLOCK**

Type	:Software accessible time of day and calendar clock. Totally independant of microprocessor.
Accuracy	:200ppm worst case, typically 2 sec per day.
Integrity	:Non volatile, battery backed.

9.1.10 **MICROPROCESSOR**

Type	:NSC800-4
Software	:Executes the Z-80 instruction set (8080 super set); can run standard CP/M compatible programs.
Cycle time	:250ns (4MHz)

9.1.11 **BATTERIES**

Main :Four 'AA' cells, Ni/Cd rechargeable for daily use. Mallory MN1500 for long shelf life and reliability.Typical operating lifetime (software dependant) 14h (Ni/Cd) 45h alkaline.

Secondary :Built-in Ni/Cd charged from main cells when operating. Life 50h when main batteries are discharged.

HUSKY Computers reserve the right to alter the specification or conditions of any product or service without prior notice.

9.2

ASCII CHARACTER SET

MSD / LSD		0 000		1 001		2 010		3 011		4 100		5 101		6 110		7 111	
		Char	Dec	Char	Dec	Char	Dec	Char	Dec	Char	Dec	Char	Dec	Char	Dec	Char	Dec
0	0000	NUL	0	DLE	16	SP	32	0	48	@	64	P	80	`	96	p	112
1	0001	SOH	1	DC1	17	!	33	1	49	A	65	Q	81	a	97	q	113
2	0010	STX	2	DC2	18	"	34	2	50	B	66	R	82	b	98	r	114
3	0011	ETX	3	DC3	19	#	35	3	51	C	67	S	83	c	99	s	115
4	0100	EOT	4	DC4	20	$	36	4	52	D	68	T	84	d	100	t	116
5	0101	ENQ	5	NAK	21	%	37	5	53	E	69	U	85	e	101	u	117
6	0110	ACK	6	SYN	22	&	38	6	54	F	70	V	86	f	102	v	118
7	0111	BEL	7	ETB	23	'	39	7	55	G	71	W	87	g	103	w	119
8	1000	BS	8	CAN	24	(	40	8	56	H	72	X	88	h	104	x	120
9	1001	HT	9	EM	25	)	41	9	57	I	73	Y	89	i	105	y	121
A	1010	LF	10	SUB	26	*	42	:	58	J	74	Z	90	j	106	z	122
B	1011	VT	11	ESC	27	+	43	;	59	K	75	[	91	k	107	{	123
C	1100	FF	12	FS	28	,	44	<	60	L	76	\	92	l	108	|	124
D	1101	CR	13	GS	29	-	45	=	61	M	77	]	93	m	109	}	125
E	1110	SO	14	RS	30	.	46	>	62	N	78	↑	94	n	110	~	126
F	1111	SI	15	US	31	/	47	?	63	O	79	←	95	o	111	DEL	127

9.3

HEX TO DECIMAL CONVERSION

The following is a conversion from Hex to Decimal for all one byte values

Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex
0	00	32	20	64	40	96	60	128	80	160	A0	192	C0	224	E0
1	01	33	21	65	41	97	61	129	81	161	A1	193	C1	225	E1
2	02	34	22	66	42	98	62	130	82	162	A2	194	C2	226	E2
3	03	35	23	67	43	99	63	131	83	163	A3	195	C3	227	E3
4	04	36	24	68	44	100	64	132	84	164	A4	196	C4	228	E4
5	05	37	25	69	45	101	65	133	85	165	A5	197	C5	229	E5
6	06	38	26	70	46	102	66	134	86	166	A6	198	C6	230	E6
7	07	39	27	71	47	103	67	135	87	167	A7	199	C7	231	E7
8	08	40	28	72	48	104	68	136	88	168	A8	200	C8	232	E8
9	09	41	29	73	49	105	69	137	89	169	A9	201	C9	233	E9
10	0A	42	2A	74	4A	106	6A	138	8A	170	AA	202	CA	234	EA
11	0B	43	2B	75	4B	107	6B	139	8B	171	AB	203	CB	235	EB
12	0C	44	2C	76	4C	108	6C	140	8C	172	AC	204	CC	236	EC
13	0D	45	2D	77	4D	109	6D	141	8D	173	AD	205	CD	237	ED
14	0E	46	2E	78	4E	110	6E	142	8E	174	AE	206	CE	238	EE
15	0F	47	2F	79	4F	111	6F	143	8F	175	AF	207	CF	239	EF
16	10	48	30	80	50	112	70	144	90	176	B0	208	D0	240	F0
17	11	49	31	81	51	113	71	145	91	177	B1	209	D1	241	F1
18	12	50	32	82	52	114	72	146	92	178	B2	210	D2	242	F2
19	13	51	33	83	53	115	73	147	93	179	B3	211	D3	243	F3
20	14	52	34	84	54	116	74	148	94	180	B4	212	D4	244	F4
21	15	53	35	85	55	117	75	149	95	181	B5	213	D5	245	F5
22	16	54	36	86	56	118	76	150	96	182	B6	214	D6	246	F6
23	17	55	37	87	57	119	77	151	97	183	B7	215	D7	247	F7
24	18	56	38	88	58	120	78	152	98	184	B8	216	D8	248	F8
25	19	57	39	89	59	121	79	153	99	185	B9	217	D9	249	F9
26	1A	58	3A	90	5A	122	7A	154	9A	186	BA	218	DA	250	FA
27	1B	59	3B	91	5B	123	7B	155	9B	187	BB	219	DB	251	FB
28	1C	60	3C	92	5C	124	7C	156	9C	188	BC	220	DC	252	FC
29	1D	61	3D	93	5D	125	7D	157	9D	189	BD	221	DD	253	FD
30	1E	62	3E	94	5E	126	7E	158	9E	190	BE	222	DE	254	FE
31	1F	63	3F	95	5F	127	7F	159	9F	191	BF	223	DF	255	FF

9.4

NSC800 MACHINE CODE

INSTRUCTION	C	Z	P/V	S	N	H	COMMENTS
ADD A, s ADC A, s	↕	↕	V	↕	0	↕	8-bit add or add with carry
SUB s SBC A, s CP s NEG	↕	↕	V	↕	1	↕	8-bit subtract subtract with carry compare and negate accumulator
AND s	0	↕	P	↕	0	1	Logical operations
OR s XOR s	0	↕	P	↕	0	0	And set different flags
INC s	•	↕	V	↕	0	•	8-bit increment
DEC m	•	↕	V	↕	1	•	8-bit decrement
ADD DD, ss	↕	•	•	•	0	X	16-bit add
ADC HL, ss	↕	↕	V	↕	0	X	16-bit add with carry
SBC HL, ss	↕	↕	V	↕	1	X	16-bit subtract with carry
RLA RLCA RRA RRCA	↕	•	•	•	0	0	Rotate accumulator
RL m RLC m RR m RRC m SLA m SRA m SRL m	↕	↕	P	↕	0	0	Rotate and shift location s
RLD RRD	•	↕	P	↕	0	0	Rotate digit left and right
DAA	↕	↕	P	↕	•	↕	Decimal adjust accumulator
CPL	•	•	•	•	1	1	Complement accumulator
SCF	1	•	•	•	0	0	Set carry
CCF	↕	•	•	•	0	X	Complement carry
IN r (C)	•	↕	P	↕	0	0	Input register indirect
INI IND OUTI OUTD	•	↕	X	X	1	X	Block input and output
INIR INDR OTIR OTDR	•	1	X	X	1	X	Z = 0 if B ≠ 0 otherwise Z = 1
LDI LDD	•	X		X	0	0	Block transfer instructions
LDIR LDDR	•	X	0	X	0	0	P/V = 1 if BC ≠ 0 otherwise P/V = 0
CPI CPIR CPD CPDR	•	↕		X	1	X	Block search instructions Z = 1 if A = (HL) otherwise Z = 0 P/V = 1 if BC ≠ 0 otherwise P/V = 0
LD A, I LD A, R	•	↕	IFF	↕	0	0	The content of the interrupt enable flip flop (IFF) is copied into the P/V flag
BIT b, s	•	↕	X	X	0	1	The complement of bit b of location s is copied into the Z flag
NEG	↕	↕	V	↕	1	↕	Negate accumulator

The following notation is used in this table

SYMBOL	OPERATION
C	Carry/link flag. C=1 if the operation produced a carry from the MSB of the operand or result
Z	Zero flag. Z=1 if the result of the operation is zero
S	Sign flag. S=1 if the MSB of the result is one
P/V	Parity or overflow flag. Parity (P) and overflow (V) share the same flag. Logical operations affect this flag with the parity of the result while arithmetic operations affect this flag with the overflow of the result. If P/V holds parity, P/V=1 if the result of the operation is even, P/V=0 if result is odd. If P/V holds overflow, P/V=1 if the result of the operation produced an overflow
H	Half-carry flag. H=1 if the add or subtract operation produced a carry into or borrow from bit 4 of the accumulator
N	Add/Subtract flag. N=1 if the previous operation was a subtract
	H and N flags are used in conjunction with the decimal adjust instruction (DAA) to properly correct the result into packed BCD format following addition or subtraction using operands with packed BCD format
↕	The flag is effected according to the result of the operation
•	The flag is unchanged by the operation
0	The flag is reset by the operation
1	The flag is set by the operation
X	The flag is a "don't care"
V	P/V flag effected according to the overflow result of the operation
P	P/V flag effected according to the parity result of the operation
r	Any one of the CPU registers A, B, C, D, E, H, L
s	Any 8-bit location for all the addressing modes allowed for the particular instruction
ss	Any 16-bit location for all the addressing modes allowed for that instruction
ii	Any one of the two index registers IX or IY
R	Refresh counter
n	8-bit value in range 0 to 255
nn	16-bit value in range 0 to 65535
m	Any 8-bit location for all the addressing modes allowed for the particular instruction

SUMMARY OF FLAG OPERATION

S	Z	X	H	X	P/V	N	C

Sequence of flags in F register

MNEMONIC	SYMBOLIC OPERATION	FLAGS C	P/V	S	N	H	OP CODE 76 543 210	NO OF T CYCLES	COMMENTS
LD r,r'	r ← r'	•	•	•	•	•	01 r r'	4	r, r' Reg: 000 B, 001 C, 010 D, 011 E, 100 H, 101 L, 111 A
LD r,n	r ← n	•	•	•	•	•	00 r 110 / n	7	
LD r,(HL)	r ← (HL)	•	•	•	•	•	01 r 110	7	
LD r,(IX+d)	r ← (IX+d)	•	•	•	•	•	11 011 101 / 01 r 110 / d	19	
LD r,(IY+d)	r ← (IY+d)	•	•	•	•	•	11 111 101 / 01 r 110 / d	19	
LD (HL),r	(HL) ← r	•	•	•	•	•	01 110 r	7	
LD (IX+d),r	(IX+d) ← r	•	•	•	•	•	11 011 101 / 01 110 r / d	19	
LD (IY+d),r	(IY+d) ← r	•	•	•	•	•	11 111 101 / 01 110 r / d	19	
LD (HL),n	(HL) ← n	•	•	•	•	•	00 110 110 / n	10	
LD (IX+d),n	(IX+d) ← n	•	•	•	•	•	11 011 101 / 00 110 110 / d / n	19	
LD (IY+d),n	(IY+d) ← n	•	•	•	•	•	11 111 101 / 00 110 110 / d / n	19	
LD A,(BC)	A ← (BC)	•	•	•	•	•	00 001 010	7	
LD A,(DE)	A ← (DE)	•	•	•	•	•	00 011 010	7	
LD A,(nn)	A ← (nn)	•	•	•	•	•	00 111 010 / n / n	13	
LD (BC),A	(BC) ← A	•	•	•	•	•	00 000 010	7	
LD (DE),A	(DE) ← A	•	•	•	•	•	00 010 010	7	
LD (nn),A	(nn) ← A	•	•	•	•	•	00 110 010 / n / n	13	
LD A,I	A ← I	•		IFF		0	0	11 101 101 / 01 010 111	9	
LD A,R	A ← R	•		IFF		0	0	11 101 101 / 01 011 111	9	
LD I,A	I ← A	•	•	•	•	•	•	11 101 101 / 01 000 111	9	
LD R,A	R ← A	•	•	•	•	•	•	11 101 101 / 01 001 111	9	
LD dd,nn	dd ← nn	•	•	•	•	•	•	00 dd0 001 / n / n	10	dd: 00 BC, 01 DE, 10 HL, 11 SP
LD IX,nn	IX ← nn	•	•	•	•	•	•	11 011 101 / 00 100 001 / n / n	14	
LD IY,nn	IY ← nn	•	•	•	•	•	•	11 111 101 / 00 100 001 / n / n	14	
LD HL,(nn)	H ← (nn+1) L ← (nn)	•	•	•	•	•	•	00 101 010 / n / n	16	
LD dd,(nn)	dd_H ← (nn+1) dd_L ← (nn)	•	•	•	•	•	•	11 101 101 / 01 dd1 011 / n / n	20	
LD IX,(nn)	IX_H ← (nn+1) IX_L ← (nn)	•	•	•	•	•	•	11 011 101 / 00 101 010 / n / n	20	
LD IY,(nn)	IY_H ← (nn+1) IY_L ← (nn)	•	•	•	•	•	•	11 111 101 / 00 101 010 / n / n	20	
LD (nn),HL	(nn+1) ← H (nn) ← L	•	•	•	•	•	•	00 100 010 / n / n	16	

MNEMONIC	SYMBOLIC OPERATION	FLAGS C	Z	P/V	S	N	H	OP CODE 76 543 210	NO OF T CYCLES	COMMENTS
LD (nn),dd	(nn+1) ← dd_H (nn) ← dd_L	•	•	•	•	•	•	11 101 101 / 01 dd0 011 / n / n	20	
LD (nn),IX	(nn+1) ← IX_H (nn) ← IX_L	•	•	•	•	•	•	11 011 101 / 00 100 010 / n / n	20	
LD (nn),IY	(nn+1) ← IY_H (nn) ← IY_L	•	•	•	•	•	•	11 111 101 / 00 100 010 / n / n	20	
LD SP,HL	SP ← HL	•	•	•	•	•	•	11 111 001	6	
LD SP,IX	SP ← IX	•	•	•	•	•	•	11 011 101 / 11 111 001	10	
LD SP,IY	SP ← IY	•	•	•	•	•	•	11 111 101 / 11 111 001	10	
PUSH qq	(SP-2) ← qq_L (SP-1) ← qq_H	•	•	•	•	•	•	11 qq0 101	11	qq: 00 BC, 01 DE, 10 HL, 11 AF
PUSH IX	(SP-2) ← IX_L (SP-1) ← IX_H	•	•	•	•	•	•	11 011 101 / 11 100 101	15	
PUSH IY	(SP-2) ← IY_L (SP-1) ← IY_H	•	•	•	•	•	•	11 111 101 / 11 100 101	15	
POP qq	qq_H ← (SP+1) qq_L ← (SP)	•	•	•	•	•	•	11 qq0 001	10	
POP IX	IX_H ← (SP+1) IX_L ← (SP)	•	•	•	•	•	•	11 011 101 / 11 100 001	14	
POP IY	IY_H ← (SP+1) IY_L ← (SP)	•	•	•	•	•	•	11 111 101 / 11 100 001	14	
EX DE,HL	DE ↔ HL	•	•	•	•	•	•	11 101 011	4	
EX AF,AF'	AF ↔ AF'	•	•	•	•	•	•	00 001 000	4	
EXX	BC ↔ BC' DE ↔ DE' HL ↔ HL'	•	•	•	•	•	•	11 011 001	4	Register bank and auxiliary register bank exchange
EX (SP),HL	H ↔ (SP+1) L ↔ (SP)	•	•	•	•	•	•	11 100 011	19	
EX (SP),IX	IX_H ↔ (SP+1) IX_L ↔ (SP)	•	•	•	•	•	•	11 011 101 / 11 100 011	23	
EX (SP),IY	IY_H ↔ (SP+1) IY_L ↔ (SP)	•	•	•	•	•	•	11 111 101 / 11 100 011	23	
LDI	(DE) ← (HL) DE ← DE+1 HL ← HL+1 BC ← BC-1	•	•	①	•	0	0	11 101 101 / 10 100 000	16	Load (HL) into (DE), increment the pointers and decrement the byte counter (BC)
LDIR	(DE) ← (HL) DE ← DE+1 HL ← HL+1 BC ← BC-1 Repeat until BC = 0	•	•	①	•	0	0	11 101 101 / 10 110 000	21 / 16	If BC ≠ 0 / If BC = 0
LDD	(DE) ← (HL) DE ← DE-1 HL ← HL-1 BC ← BC-1	•	•	①	•	0	0	11 101 101 / 10 101 000	16	
LDDR	(DE) ← (HL) DE ← DE-1 HL ← HL-1 BC ← BC-1 Repeat until BC = 0	•	•	①	•	0	0	11 101 101 / 10 111 000	21 / 16	If BC ≠ 0 / If BC = 0
CPI	A - (HL) HL ← HL+1 BC ← BC-1	•	②	①		1		11 101 101 / 10 100 001	16	
CPIR	A - (HL) HL ← HL-1 BC ← BC-1 Repeat until A = (HL) or BC = 0	•	②	①		1		11 101 101 / 10 110 001	21 / 16	If BC ≠ 0 and A ≠ (HL) / If BC = 0 or A = (HL)
CPD	A - (HL) HL ← HL-1 BC ← BC-1	•	②	①		1		11 101 101 / 10 101 001	16	

MNEMONIC	SYMBOLIC OPERATION	FLAGS C	Z	P/V	S	N	H	OP CODE 76 543 210	NO. OF T CYCLES	COMMENTS
CPDR	A (HL) HL HL 1 BC BC 1 Repeat until A (HL) or BC = 0	•	↕	↕		1	↕	11 101 101 10 111 001	21 16	1 BC = 0 and A ≠ (HL) If BC = 0 or A = (HL)
ADD A, r	A A + r	↕	↕	V	↕	0	↕	10 000 r	4	r | Reg 000 B 001 C 010 D 011 E 100 H 101 L 111 A
ADD A, n	A A + n	↕	↕	V	↕	0	↕	11 000 110 n	7	
ADD A, (HL)	A A + (HL)	↕	↕	V	↕	0	↕	10 000 110	7	
ADD A, (IX+d)	A A + (IX+d)	↕	↕	V	↕	0	↕	11 011 101 10 000 110 d	19	
ADD A, (IY+d)	A A + (IY+d)	↕	↕	V	↕	0	↕	11 111 101 10 000 110 d	19	
ADC A, s	A A + s + CY	↕	↕	V	↕	0	↕	001		s = any of r, n (HL), (IX+d) (IY+d) as shown for ADD instruction
SUB s	A A - s	↕	↕	V	↕	1	↕	010		
SBC A, s	A A - s - CY	↕	↕	V	↕	1	↕	011		
AND s	A A ∧ s	0	↕	P	↕	1	1	100		The indicated bits replace the 000 in the ADD set above
OR s	A A ∨ s	0	↕	P	↕	0	0	110		
XOR s	A A ⊕ s	0	↕	P	↕	0	0	101		
CP s	A - s	↕	↕	V	↕	1	↕	111		
INC r	r r + 1	•	↕	V	↕	0	↕	00 r 100	4	
INC (HL)	(HL) (HL) + 1	•	↕	V	↕	0	↕	00 110 100	11	
INC (IX+d)	(IX+d) (IX+d) + 1	•	↕	V	↕	0	↕	11 011 101 00 110 100 d	23	
INC (IY+d)	(IY+d) (IY+d) + 1	•	↕	V	↕	0	↕	11 111 101 00 110 100 d	23	
DEC m	m m - 1	•	↕	V	↕	1	↕	101		m = any of r, (HL) or (IX+d), (IY+d) as shown for INC Same format and states as INC Replace 100 with 101 in OP code
ADD HL, ss	HL HL + ss	↕	•	•	•	0	X	00 ss1 001	11	ss | Reg 00 BC 01 DE 10 HL 11 SP
ADC HL, ss	HL HL + ss + CY	↕	↕	V	↕	0	X	11 101 101 01 ss1 010	15	
SBC HL, ss	HL HL - ss - CY	↕	↕	V	↕	1	X	11 101 101 01 ss0 010	15	
ADD IX, pp	IX IX + pp	•	•	•	•	0	X	11 011 101 00 pp1 001	15	pp | Reg 00 BC 01 DE 10 IX 11 SP
ADD IY, rr	IY IY + rr	↕	•	•	•	0	X	11 111 101 00 rr1 001	15	rr | Reg 00 BC 01 DE 10 IY 11 SP
INC ss	ss ss + 1	•	•	•	•	•	•	00 ss0 011	6	
INC IX	IX IX + 1	•	•	•	•	•	•	11 011 101 00 100 011	10	
INC IY	IY IY + 1	•	•	•	•	•	•	11 111 101 00 100 011	10	
DEC ss	ss ss - 1	•	•	•	•	•	•	00 ss1 011	6	
DEC IX	IX IX - 1	•	•	•	•	•	•	11 011 101 00 101 011	10	
DEC IY	IY IY - 1	•	•	•	•	•	•	11 111 101 00 101 011	10	
RLCA		↕	•	•	•	0	0	00 000 111	4	Rotate left circular accumulator
RLA		↕	•	•	•	0	0	00 010 111	4	Rotate left accumulator
RRCA		↕	•	•	•	0	0	00 001 111	4	Rotate right circular accumulator
RRA		↕	•	•	•	0	0	00 011 111	4	Rotate right accumulator

MNEMONIC	SYMBOLIC OPERATION	FLAGS C	Z	P/V	S	N	H	OP CODE 76 543 210	NO. OF T CYCLES	COMMENTS
RLC r		↕	↕	P	↕	0	0	11 001 011 00 000 r	8	Rotate left circular register r
RLC (HL)		↕	↕	P	↕	0	0	11 001 011 00 000 110	15	r | Reg 000 B 001 C 010 D 011 E 100 H 101 L 111 A
RLC (IX+d)		↕	↕	P	↕	0	0	11 011 101 11 001 011 d 00 000 110	23	
RLC (IY+d)		↕	↕	P	↕	0	0	11 111 101 11 001 011 d 00 000 110	23	
RL m		↕	↕	P	↕	0	0	010		Instruction format and states are as shown for RLC. To form new OP code replace 000 for RLC with shown code
RRC m		↕	↕	P	↕	0	0	001		
RR m		↕	↕	P	↕	0	0	011		
SLA m		↕	↕	P	↕	0	0	100		
SRA m		↕	↕	P	↕	0	0	101		
SRL m		↕	↕	P	↕	0	0	111		
RLD		•	↕	P	↕	0	0	11 101 101 01 101 111	18	Rotate digits left and right between the accumulator and location (HL). The contents of the upper half of the accumulator are unaffected
RRD		•	↕	P	↕	0	0	11 101 101 01 100 111	18	
DAA	Converts acc. content into packed BCD following add or subtract with packed BCD operands	↕	↕	P	↕	•	↕	00 100 111	4	Decimal adjust accumulator
CPL	A A	•	•	•	•	1	1	00 101 111	4	Complement accumulator (one's complement)
NEG	A 0 - A	↕	↕	V	↕	1	↕	11 101 101 01 000 100	8	Negate acc. (two's complement)
CCF	CY CY	↕	•	•	•	0	↕	00 111 111	4	Complement carry flag
SCF	CY 1	1	•	•	•	0	0	00 110 111	4	Set carry flag
NOP	No operation	•	•	•	•	•	•	00 000 000	4	
HALT	CPU halted	•	•	•	•	•	•	01 110 110	4	
DI	IFF 0	•	•	•	•	•	•	11 110 011	4	
EI	IFF 1	•	•	•	•	•	•	11 111 011	4	
IM 0	Set interrupt mode 0	•	•	•	•	•	•	11 101 101 01 000 110	8	
IM 1	Set interrupt mode 1	•	•	•	•	•	•	11 101 101 01 010 110	8	
IM 2	Set interrupt mode 2	•	•	•	•	•	•	11 101 101 01 011 110	8	
BIT b, r	Z r_b	•	↕	X	X	0	1	11 001 011 01 b r	8	b | Bit 000 0 001 1 010 2 011 3 100 4 101 5 110 6 111 7 r | Reg 000 B 001 C 010 D 011 E 100 H 101 L 111 A
BIT b, (HL)	Z (HL)_b	•	↕	X	X	0	1	11 001 011 01 b 110	12	
BIT b, (IX+d)	Z (IX+d)_b	•	↕	X	X	0	1	11 011 101 11 001 011 d 01 b 110	20	
BIT b, (IY+d)	Z (IY+d)_b	•	↕	X	X	0	1	11 111 101 11 001 011 d 01 b 110	20	

MNEMONIC	SYMBOLIC OPERATION	FLAGS C	Z	P/V	S	N	H	OP-CODE 76 543 210	NO OF T CYCLES	COMMENTS
SET b, r	$r_b \leftarrow 1$	•	•	•	•	•	•	11 001 011 11 b r	8	
SET b, (HL)	$(HL)_b \leftarrow 1$	•	•	•	•	•	•	11 001 011 11 b 110	15	
SET b, (IX+d)	$(IX+d)_b \leftarrow 1$	•	•	•	•	•	•	11 011 101 11 001 011 — d — 11 b 110	23	
SET b, (IY+d)	$(IY+d)_b \leftarrow 1$	•	•	•	•	•	•	11 111 101 11 001 011 — d — 11 b 110	23	
RES b, m	$m_b \leftarrow 0$ m = (HL) (IX+d) (IY+d)							10		To form new OP code replace 11 of SET b, r with 10. Flags and time states for SET instruction
JP nn	PC ← nn	•	•	•	•	•	•	11 000 011 — n — — n —	10	
JP cc, nn	If condition is true PC ← nn otherwise continue	•	•	•	•	•	•	11 cc 010 — n — — n —	10	cc | Condition 000 | NZ non zero 001 | Z zero 010 | NC non carry 011 | C carry 100 | PO parity odd 101 | PE parity even 110 | P sign positive 111 | M sign negative
JR e	PC ← PC + e	•	•	•	•	•	•	00 011 000 — e-2 —	12	
JR C, e	If C = 0 continue	•	•	•	•	•	•	00 111 000 — e-2 —	7	If condition not met
	If C = 1 PC ← PC + e								12	If condition is met
JR NC, e	If C = 1 continue	•	•	•	•	•	•	00 110 000 — e-2 —	7	If condition not met
	If C = 0 PC ← PC + e								12	If condition is met
JR Z, e	If Z = 0 continue	•	•	•	•	•	•	00 101 000 — e-2 —	7	If condition not met
	If Z = 1 PC ← PC + e								12	If condition is met
JR NZ, e	If Z = 1 continue	•	•	•	•	•	•	00 100 000 — e-2 —	7	If condition not met
	If Z = 0 PC ← PC + e								12	If condition met
JP (HL)	PC ← HL	•	•	•	•	•	•	11 101 001	4	
JP (IX)	PC ← IX	•	•	•	•	•	•	11 011 101 11 101 001	8	
JP (IY)	PC ← IY	•	•	•	•	•	•	11 111 101 11 101 001	8	
DJNZ, e	B ← B-1 If B = 0 continue	•	•	•	•	•	•	00 010 000 — e-2 —	8	If B = 0
	If B ≠ 0 PC ← PC + e								13	If B ≠ 0
CALL nn	(SP-1) ← PC_H (SP-2) ← PC_L PC ← nn	•	•	•	•	•	•	11 001 101 — n — — n —	17	
CALL cc, nn	If condition cc is false continue otherwise same as CALL nn	•	•	•	•	•	•	11 cc 100 — n — — n —	10	If cc is false
									17	If cc is true
RET	$PC_L \leftarrow (SP)$ $PC_H \leftarrow (SP+1)$	•	•	•	•	•	•	11 001 001	10	
RET cc	If condition cc is false continue otherwise same as RET	•	•	•	•	•	•	11 cc 000	5	If cc is false
									11	If cc is true cc | Condition 000 | NZ non zero 001 | Z zero 010 | NC non carry 011 | C carry 100 | PO parity odd 101 | PE parity even 110 | P sign positive 111 | M sign negative
RETI	Return from interrupt	•	•	•	•	•	•	11 101 101 01 001 101	14	
RETN	Return from non maskable interrupt	•	•	•	•	•	•	11 101 101 01 000 101	14	

MNEMONIC	SYMBOLIC OPERATION	FLAGS C	Z	P/V	S	N	H	OP-CODE 76 543 210	NO OF T CYCLES	COMMENTS
RST p	(SP-1) ← PC_H (SP-2) ← PC_L PC_H ← 0 PC_L ← p	•	•	•	•	•	•	11 t 111	11	
IN A, (n)	A ← (n)	•	•	•	•	•	•	11 011 011 n	11	
IN r, (C)	r ← (C) If r = 110 only the flags will be affected	•	•	P		0		11 101 101 01 r 000	12	
INI	(HL) ← (C) B ← B-1 HL ← HL+1	•	?	X	?	1	X	11 101 101 10 100 010	16	
INIR	(HL) ← (C) B ← B-1 HL ← HL+1 Repeat until B = 0	•	1	X	1	1	X	11 101 101 10 110 010	21 16	
IND	(HL) ← (C) B ← B-1 HL ← HL-1	•	?	X	?	1	X	11 101 101 10 101 010	16	
INDR	(HL) ← (C) B ← B-1 HL ← HL-1 Repeat until B = 0	•	1	X	1	1	X	11 101 101 10 111 010	21 16	
OUT (n), A	(n) ← A	•	•	•	•	•	•	11 010 011 n	11	
OUT (C), r	(C) ← r	•	•	•	•	•	•	11 101 101 01 r 001	12	
OUTI	(C) ← (HL) B ← B-1 HL ← HL+1	•	?	X	?	1	X	11 101 101 10 100 011	16	
OTIR	(C) ← (HL) B ← B-1 HL ← HL+1 Repeat until B = 0	•	1	X	1	1	X	11 101 101 10 110 011	21 16	
OUTD	(C) ← (HL) B ← B-1 HL ← HL-1	•	?	X	?	1	X	11 101 101 10 101 011	16	
OTDR	(C) ← (HL) B ← B-1 HL ← HL-1 Repeat until B = 0	•	1	X	1	1	X	11 101 101 10 111 011	21 16	

Notes: r, r' means any of the registers A, B, C, D, E, H, L
ss is any of the register pairs BC, DE, HL, SP
qq is any of the register pairs BC, DE, HL, AF
1. P/V flag = 1 if the result of B - 1 = 0, otherwise P/V = 0
2. Z flag = 1 if A = (HL), otherwise Z = 0
3. If the result of B - 1 ≠ 0, the Z flag is set, otherwise it is reset
d represents the extension in the location addressing mode
e is a signed two's complement number in the range -126, 129
e-2 in the op code provides an effective address of pc + e as PC is incremented before addition occurs
The notation n_b indicates bit b (0 to 7) of location n

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag unknown
? = flag affected according to the result of the operation

OBJ CODE	SOURCE STATEMENT		OPERATION
8E	ADC	A,(HL)	Add with Carry Oper
DD8E05	ADC	A,(IX+d)	and to Acc
FD8E05	ADC	A,(IY+d)	
8F	ADC	A,A	
88	ADC	A,B	
89	ADC	A,C	
8A	ADC	A,D	
8B	ADC	A,E	
8C	ADC	A,H	
8D	ADC	A,L	
CE20	ADC	A,n	
ED4A	ADC	HL,BC	Add with Carry Reg
ED5A	ADC	HL,DE	Pair to HL
ED6A	ADC	HL,HL	
ED7A	ADC	HL,SP	
86	ADD	A,(HL)	Add Operand to Acc
DD8605	ADD	A,(IX+d)	
FD8605	ADD	A,(IY+d)	
87	ADD	A,A	
80	ADD	A,B	
81	ADD	A,C	
82	ADD	A,D	
83	ADD	A,E	
84	ADD	A,H	
85	ADD	A,L	
C620	ADD	A,n	
09	ADD	HL,BC	Add Reg Pair to HL
19	ADD	HL,DE	
29	ADD	HL,HL	
39	ADD	HL,SP	
DD09	ADD	IX,BC	Add Reg Pair to IX
DD19	ADD	IX,DE	
DD29	ADD	IX,IX	
DD39	ADD	IX,SP	
FD09	ADD	IY,BC	Add Reg Pair to IY
FD19	ADD	IY,DE	
FD29	ADD	IY,IY	
FD39	ADD	IY,SP	
A6	AND	(HL)	Logical 'AND' of
DDA605	AND	(IX+d)	Operand and Acc
FDA605	AND	(IY+d)	
A7	AND	A	
A0	AND	B	
A1	AND	C	
A2	AND	D	
A3	AND	E	
A4	AND	H	
A5	AND	L	
E620	AND	n	
CB46	BIT	0,(HL)	Test Bit b of Location
DDCB0546	BIT	0,(IX+d)	or Reg
FDCB0546	BIT	0,(IY+d)	
CB47	BIT	0,A	
CB40	BIT	0,B	
CB41	BIT	0,C	
CB42	BIT	0,D	
CB43	BIT	0,E	
CB44	BIT	0,H	

OBJ CODE	SOURCE STATEMENT		OPERATION
CB45	BIT	0,L	Test Bit b of Location
CB4E	BIT	1,(HL)	or Reg
DDCB054E	BIT	1,(IX+d)	
FDCB054E	BIT	1,(IY+d)	
CB4F	BIT	1,A	
CB48	BIT	1,B	
CB49	BIT	1,C	
CB4A	BIT	1,D	
CB4B	BIT	1,E	
CB4C	BIT	1,H	
CB4D	BIT	1,L	
CB56	BIT	2,(HL)	
DDCB0556	BIT	2,(IX+d)	
FDCB0556	BIT	2,(IY+d)	
CB57	BIT	2,A	
CB50	BIT	2,B	
CB51	BIT	2,C	
CB52	BIT	2,D	
CB53	BIT	2,E	
CB54	BIT	2,H	
CB55	BIT	2,L	
CB5E	BIT	3,(HL)	
DDCB055E	BIT	3,(IX+d)	
FDCB055E	BIT	3,(IY+d)	
CB5F	BIT	3,A	
CB58	BIT	3,B	
CB59	BIT	3,C	
CB5A	BIT	3,D	
CB5B	BIT	3,E	
CB5C	BIT	3,H	
CB5D	BIT	3,L	
CB66	BIT	4,(HL)	
DDCB0566	BIT	4,(IX+d)	
FDCB0566	BIT	4,(IY+d)	
CB67	BIT	4,A	
CB60	BIT	4,B	
CB61	BIT	4,C	
CB62	BIT	4,D	
CB63	BIT	4,E	
CB64	BIT	4,H	
CB65	BIT	4,L	
CB6E	BIT	5,(HL)	
DDCB056E	BIT	5,(IX+d)	
FDCB056E	BIT	5,(IY+d)	
CB6F	BIT	5,A	
CB68	BIT	5,B	
CB69	BIT	5,C	
CB6A	BIT	5,D	
CB6B	BIT	5,E	
CB6C	BIT	5,H	
CB6D	BIT	5,L	
CB76	BIT	6,(HL)	
DDCB0576	BIT	6,(IX+d)	
FDCB0576	BIT	6,(IY+d)	
CB77	BIT	6,A	
CB70	BIT	6,B	
CB71	BIT	6,C	
CB72	BIT	6,D	
CB73	BIT	6,E	
CB74	BIT	6,H	
CB75	BIT	6,L	
CB7E	BIT	7,(HL)	
DDCB057E	BIT	7,(IX+d)	

OBJ CODE	SOURCE STATEMENT		OPERATION
FDCB057E	BIT	7,(IY+d)	Test Bit b Location or Reg
CB7F	BIT	7,A	
CB78	BIT	7,B	
CB79	BIT	7,C	
CB7A	BIT	7,D	
CB7B	BIT	7,E	
CB7C	BIT	7,H	
CB7D	BIT	7,L	
DC8405	CALL	C,nn	Call Subroutine at Location nn if Condition True
FC8405	CALL	M,nn	
D48405	CALL	NC,nn	
C48405	CALL	NZ,nn	
F48405	CALL	P,nn	
EC8405	CALL	PE,nn	
E48405	CALL	PO,nn	
CC8405	CALL	Z,nn	
CD8405	CALL	nn	Unconditional Call to Subroutine at nn
3F	CCF		Complement Carry Flag
BE	CP	(HL)	Compare Operand with Acc
DDBE05	CP	(IX+d)	
FDBE05	CP	(IY+d)	
BF	CP	A	
B8	CP	B	
B9	CP	C	
BA	CP	D	
BB	CP	E	
BC	CP	H	
BD	CP	L	
FE20	CP	n	
EDA9	CPD		Compare Location (HL) and Acc Decrement HL and BC
EDB9	CPDR		Compare Location (HL) and Acc Decrement HL and BC, Repeat until BC = 0
EDA1	CPI		Compare Location (HL) and Acc Increment HL and Decrement BC
EDB1	CPIR		Compare Location (HL) and Acc Increment HL, Decrement BC, Repeat until BC = 0
2F	CPL		Complement Acc (1's Compl
27	DAA		Decimal Adjust Acc
35	DEC	(HL)	Decrement Operand
DD3505	DEC	(IX+d)	
FD3505	DEC	(IY+d)	
3D	DEC	A	
05	DEC	B	
08	DEC	BC	
0D	DEC	C	
15	DEC	D	
1B	DEC	DE	

OBJ CODE	SOURCE STATEMENT		OPERATION
1D	DEC	E	Decrement Operand
25	DEC	H	
2B	DEC	HL	
DD2B	DEC	IX	
FD2B	DEC	IY	
2D	DEC	L	
3B	DEC	SP	
F3	DI		Disable Interrupts
102E	DJNZ	e	Decrement B and Jump Relative if B ≠ 0
FB	EI		Enable Interrupts
E3	EX	(SP),HL	Exchange Location and (SP)
DDE3	EX	(SP),IX	
FDE3	EX	(SP),IY	
08	EX	AF,AF'	Exchange the Contents of AF and AF'
EB	EX	DE,HL	Exchange the Contents of DE and HL
D9	EXX		Exchange the Contents of BC, DE, HL with Contents of BC', DE', HL' Respectively
76	HALT		HALT (Wait for Interrupt or Reset)
ED46	IM	0	Set Interrupt Mode
ED56	IM	1	
ED5E	IM	2	
ED78	IN	A,(C)	Load Reg. with Input from Device (C)
ED40	IN	B,(C)	
ED48	IN	C,(C)	
ED50	IN	D,(C)	
ED58	IN	E,(C)	
ED60	IN	H,(C)	
ED68	IN	L,(C)	
34	INC	(HL)	Increment Operand
DD3405	INC	(IX+d)	
FD3405	INC	(IY+d)	
3C	INC	A	
04	INC	B	
03	INC	BC	
0C	INC	C	
14	INC	D	
13	INC	DE	
1C	INC	E	
24	INC	H	
23	INC	HL	
DD23	INC	IX	
FD23	INC	IY	
2C	INC	L	
33	INC	SP	
DB20	IN	A,(n)	Load Acc with Input from Device (n)
EDAA	IND		Load Location (HL) with Input from Port (C), Decrement HL and B

OBJ CODE	SOURCE STATEMENT		OPRRTION
EDBA	INDR		Load Location (HL) with Input from Port (C) Decrement HL and Decrement B Repeat until B = 0
EDA2	INI		Load Location (HL) with Input from Port (C) Increment HL and Decrement B
EDB2	INIR		Load Location (HL) with Input from Port (C) Increment HL and Decrement B Repeat until B = 0
E9	JP	(HL)	Unconditional Jump to Location
DDE9	JP	(IX)	
C38405	JP	nn	
FDE9	JP	(IY)	
DAB405	JP	C,nn	Jump to Location if Condition True
FAB405	JP	M,nn	
D2B405	JP	NC,nn	
C2B405	JP	NZ,nn	
F2B405	JP	P,nn	
EAB405	JP	PE,nn	
E2B405	JP	PO,nn	
CAB405	JP	Z,nn	
382E	JR	C,e	Jump Relative to PC+e if Condition
302E	JR	NC,e	True
202E	JR	NZ,e	
282E	JR	Z,e	
182E	JR	e	Unconditional Jump Relative to PC+e
02	LD	(dC),A	Load Source to Des
12	LD	(DE),A	tination
77	LD	(HL),A	
70	LD	(HL),B	
71	LD	(HL),C	
72	LD	(HL),D	
73	LD	(HL),E	
74	LD	(HL),H	
75	LD	(HL),L	
3620	LD	(HL),n	
DD7705	LD	(IX+d),A	
DD7005	LD	(IX+d),B	
DD7105	LD	(IX+d),C	
DD7205	LD	(IX+d),D	
DD7305	LD	(IX+d),E	
DD7405	LD	(IX+d),H	
DD7505	LD	(IX+d),L	
DD360520	LD	(IX+d),n	
FD7705	LD	(IY+d),A	
FD7005	LD	(IY+d),B	
FD7105	LD	(IY+d),C	
FD7205	LD	(IY+d),D	
FD7305	LD	(IY+d),E	
FD7405	LD	(IY+d),H	
FD7505	LD	(IY+d),L	
FD360520	LD	(IY+d),n	
328405	LD	(nn),A	
ED438405	LD	(nn),BC	

OBJ CODE	SOURCE STATEMENT		OPERATION
ED538405	LD	(nn),DE	Load Source to Des
228405	LD	(nn),HL	tination
DD228405	LD	(nn),IX	
FD228405	LD	(nn),IY	
ED738405	LD	(nn),SP	
0A	LD	A,(BC)	
1A	LD	A,(DE)	
7E	LD	A,(HL)	
DD7E05	LD	A,(IX+d)	
FD7E05	LD	A,(IY+d)	
3A8405	LD	A,(nn)	
7F	LD	A,A	
78	LD	A,B	
79	LD	A,C	
7A	LD	A,D	
7B	LD	A,E	
7C	LD	A,H	
ED57	LD	A,I	
7D	LD	A,L	
3E20	LD	A,n	
ED5F	LD	A,R	
46	LD	B,(HL)	
DD4605	LD	B,(IX+d)	
FD4605	LD	B,(IY+d)	
47	LD	B,A	
40	LD	B,B	
41	LD	B,C	
42	LD	B,D	
43	LD	B,E	
44	LD	B,H	
45	LD	B,L	
0620	LD	B,n	
ED4B8405	LD	BC,(nn)	
018405	LD	BC,nn	
4E	LD	C,(HL)	
DD4E05	LD	C,(IX+d)	
FD4E05	LD	C,(IY+d)	
4F	LD	C,A	
48	LD	C,B	
49	LD	C,C	
4A	LD	C,D	
4B	LD	C,E	
4C	LD	C,H	
4D	LD	C,L	
0E20	LD	C,n	
56	LD	D,(HL)	
DD5605	LD	D,(IX+d)	
FD5605	LD	D,(IY+d)	
57	LD	D,A	
50	LD	D,B	
51	LD	D,C	
52	LD	D,D	
53	LD	D,E	
54	LD	D,H	
55	LD	D,L	
1620	LD	D,n	
ED5B8405	LD	DE,(nn)	
118405	LD	DE,nn	
5E	LD	E,(HL)	
DD5E05	LD	E,(IX+d)	
FD5E05	LD	E,(IY+d)	
5F	LD	E,A	
58	LD	E,B	
59	LD	E,C	

OBJ CODE	SOURCE STATEMENT		OPERATION
5A	LD	E D	Load Source to Des tination
5B	LD	E E	
5C	LD	E H	
5D	LD	E L	
1E20	LD	E n	
66	LD	H (HL)	
DD6605	LD	H (IX+d)	
FD6605	LD	H (IY+d)	
67	LD	H A	
60	LD	H B	
61	LD	H C	
62	LD	H D	
63	LD	H E	
64	LD	H H	
65	LD	H L	
2620	LD	H n	
2A8405	LD	HL (nn)	
218405	LD	HL nn	
ED47	LD	I A	
DD2A8405	LD	IX (nn)	
DD218405	LD	IX nn	
FD2A8405	LD	IY (nn)	
FD218405	LD	IY nn	
6E	LD	L (HL)	
DD6E05	LD	L (IX+d)	
FD6E05	LD	L (IY+d)	
6F	LD	L A	
68	LD	L B	
69	LD	L C	
6A	LD	L D	
6B	LD	L E	
6C	LD	L H	
6D	LD	L L	
2E20	LD	L n	
ED4F		LD R A	
ED788405	LD	SP (nn)	
F9	LD	SP HL	
DDF9	LD	SP IX	
FDF9	LD	SP IY	
318405	LD	SP nn	
EDA8	LDD		Load Location (DE) with Location (HL) Decrement DE HL and BC
EDB8	LDDR		Load Location (DE) with Location (HL) Repeat until BC = 0
EDA0	LDI		Load Location (DE) with Location (HL), Increment DE HL Decrement BC
EDB0	LDIR		Load Location (DE) with Location (HL) Increment DE HL Decrement BC and Repeat until BC = 0
ED44	NEG		Negate Acc (2's Complement)
00	NOP		No Operation
B6	OR	(HL)	Logical "OR" of
DDB605	OR	(IX+d)	Operand and Acc

OBJ CODE	SOURCE STATEMENT		OPERATION
FDB605	OR	(IY+d)	Logical "OR" of
B7	OR	A	Operand and Acc
B0	OR	B	
B1	OR	C	
B2	OR	D	
B3	OR	E	
B4	OR	H	
B5	OR	L	
F620	OR	n	
EDB8	OTDR		Load Output Port (C) with Location (HL) Decrement HL and B Repeat until B = 0
EDB3	OTIR		Load Output Port (C) with Location (HL) Increment HL, Decre ment B, Repeat until B = 0
ED79	OUT	(C),A	Load Output Port (C)
ED41	OUT	(C),B	with Reg
ED49	OUT	(C),C	
ED51	OUT	(C),D	
ED59	OUT	(C),E	
ED61	OUT	(C),H	
ED69	OUT	(C),L	
D320	OUT	(n),A	Load Output Port (n) with Acc
EDA8	OUTD		Load Output Port (C) with Location (HL), Decrement HL and B
EDA3	OUTI		Load Output Port (C) with Location (HL) Increment HL and Decrement B
F1	POP	AF	Load Destination
C1	POP	BC	with Top of Stack
D1	POP	DE	
E1	POP	HL	
DDE1	POP	IX	
FDE1	POP	IY	
F5	PUSH	AF	Load Source to Stack
C5	PUSH	BC	
D5	PUSH	DE	
E5	PUSH	HL	
DDE5	PUSH	IX	
FDE5	PUSH	IY	
CB86	RES	0,(HL)	Reset Bit b of
DDCB0586	RES	0,(IX+d)	Operand
FDCB0586	RES	0,(IY+d)	
CB87	RES	0,A	
CB80	RES	0,B	
CB81	RES	0,C	
CB82	RES	0,D	
CB83	RES	0,E	
CB84	RES	0,H	
CB85	RES	0,L	
CB8E	RES	1,(HL)	
DDCB058E	RES	1,(IX+d)	
FDCB058E	RES	1,(IY+d)	
CB8F	RES	1,A	

OBJ CODE	SOURCE STATEMENT		OPERATION
CB88	RES	1,B	Reset Bit b of Operand
CB89	RES	1,C	
CB8A	RES	1,D	
CB8B	RES	1,E	
CB8C	RES	1,H	
CB8D	RES	1,L	
CB96	RES	2,(HL)	
DDCB0596	RES	2,(IX+d)	
FDCB0596	RES	2,(IY+d)	
CB97	RES	2,A	
CB90	RES	2,B	
CB91	RES	2,C	
CB92	RES	2,D	
CB93	RES	2,E	
CB94	RES	2,H	
CB95	RES	2,L	
CB9E	RES	3,(HL)	
DDCB059E	RES	3,(IX+d)	
FDCB059E	RES	3,(IY+d)	
CB9F	RES	3,A	
CB98	RES	3,B	
CB99	RES	3,C	
CB9A	RES	3,D	
CB9B	RES	3,E	
CB9C	RES	3,H	
CB9D	RES	3,L	
CBA6	RES	4,(HL)	
DDCB05A6	RES	4,(IX+d)	
FDCB05A6	RES	4,(IY+d)	
CBA7	RES	4,A	
CBA0	RES	4,B	
CBA1	RES	4,C	
CBA2	RES	4,D	
CBA3	RES	4,E	
CBA4	RES	4,H	
CBA5	RES	4,L	
CBAE	RES	5,(HL)	
DDCB05AE	RES	5,(IX+d)	
FDCB05AE	RES	5,(IY+d)	
CBAF	RES	5,A	
CBAB	RES	5,B	
CBA9	RES	5,C	
CBAA	RES	5,D	
CBAB	RES	5,E	
CBAC	RES	5,H	
CBAD	RES	5,L	
CBB6	RES	6,(HL)	
DDCB05B6	RES	6,(IX+d)	
FDCB05B6	RES	6,(IY+d)	
CBB7	RES	6,A	
CBB0	RES	6,B	
CBB1	RES	6,C	
CBB2	RES	6,D	
CBB3	RES	6,E	
CBB4	RES	6,H	
CBB5	RES	6,L	
CBBE	RES	7,(HL)	
DDCB05BE	RES	7,(IX+d)	
FDCB05BE	RES	7,(IY+d)	
CBBF	RES	7,A	
CBB8	RES	7,B	
CBB9	RES	7,C	
CBBA	RES	7,D	

OBJ CODE	SOURCE STATEMENT		OPERATION
CBBB	RES	7,E	Reset Bit b of Operand
CBBC	RES	7,H	
CBBD	RES	7,L	
C9	RET		Return from Subroutine
D8	RET	C	Return from Subroutine if Cond. true
F8	RET	M	
D0	RET	NC	
C0	RET	NZ	
F0	RET	P	
E8	RET	PE	
E0	RET	PO	
C8	RET	Z	
ED4D	RETI		Return from Interrupt
ED45	RETN		Return from Non Maskable Interrupt
CB16	RL	(HL)	Rotate Left Through Carry
DDCB0516	RL	(IX+d)	
FDCB0516	RL	(IY+d)	
CB17	RL	A	
CB10	RL	B	
CB11	RL	C	
CB12	RL	D	
CB13	RL	E	
CB14	RL	H	
CB15	RL	L	
17	RLA		Rotate Left Acc Through Carry
CB06	RLC	(HL)	Rotate Left Circular
DDCB0506	RLC	(IX+d)	
FDCB0506	RLC	(IY+d)	
CB07	RLC	A	
CB00	RLC	B	
CB01	RLC	C	
CB02	RLC	D	
CB03	RLC	E	
CB04	RLC	H	
CB05	RLC	L	
07	RLCA		Rotate Left Circular Acc
ED6F	RLD		Rotate Digit Left and Right between Acc and and Location (HL)
CB1E	RR	(HL)	Rotate Right Through Carry
DDCB051E	RR	(IX+d)	
FDCB051E	RR	(IY+d)	
CB1F	RR	A	
CB1B	RR	B	
CB19	RR	C	
CB1A	RR	D	
CB1B	RR	E	
CB1C	RR	H	
CB1D	RR	L	
1F	RRA		Rotate Right Acc Through Carry
CB0E	RRC	(HL)	Rotate Right Circular
DDCB050E	RRC	(IX+d)	
FDCB050E	RRC	(IY+d)	
CB0F	RRC	A	

OBJ CODE	SOURCE STATEMENT		OPERATION
CB0B	RRC	B	Rotate Right Circular
CB09	RRC	C	
CB0A	RRC	D	
CB0B	RRC	E	
CB0C	RRC	H	
CB0D	RRC	L	
0F	RRCA		Rotate Right Circular Acc
ED67	RRD		Rotate Digit Right and Left Between Acc. and Location (HL)
C7	RST	00H	Restart to Location
CF	RST	08H	
D7	RST	10H	
DF	RST	18H	
E7	RST	20H	
EF	RST	28H	
F7	RST	30H	
FF	RST	38H	
9E	SBC	A,(HL)	Subtract Operand
DD9E05	SBC	A,(IX+d)	from Acc with Carry
FD9E05	SBC	A,(IY+d)	
9F	SBC	A,A	
98	SBC	A,B	
99	SBC	A,C	
9A	SBC	A,D	
9B	SBC	A,E	
9C	SBC	A,H	
9D	SBC	A,L	
DE20	SBC	A,n	
ED42	SBC	HL,BC	
ED52	SBC	HL,DE	
ED62	SBC	HL,HL	
ED72	SBC	HL,SP	
37	SCF		Set Carry Flag (C = 1)
CBC6	SET	0,(HL)	Set Bit b of Location
DDCB05C6	SET	0,(IX+d)	
FDCB05C6	SET	0,(IY+d)	
CBC7	SET	0,A	
CBC0	SET	0,B	
CBC1	SET	0,C	
CBC2	SET	0,D	
CBC3	SET	0,E	
CBC4	SET	0,H	
CBC5	SET	0,L	
CBCE	SET	1,(HL)	
DDCB05CE	SET	1,(IX+d)	
FDCB05CE	SET	1,(IY+d)	
CBCF	SET	1,A	
CBCB	SET	1,B	
CBC9	SET	1,C	
CBCA	SET	1,D	
CBCB	SET	1,E	
CBCC	SET	1,H	
CBCD	SET	1,L	
CBD6	SET	2,(HL)	
DDCB05D6	SET	2,(IX+d)	
FDCB05D6	SET	2,(IY+d)	
CBD7	SET	2,A	
CBD0	SET	2,B	
CBD1	SET	2,C	
CBD2	SET	2,D	

OBJ CODE	SOURCE STATEMENT		OPERATION
CBD3	SET	2,E	Set Bit b of Location
CBD4	SET	2,H	
CBD5	SET	2,L	
CBD8	SET	3,B	
CBDE	SET	3,(HL)	
DDCB05DE	SET	3,(IX+d)	
FDCB05DE	SET	3,(IY+d)	
CBDF	SET	3,A	
CBD9	SET	3,C	
CBDA	SET	3,D	
CBDB	SET	3,E	
CBDC	SET	3,H	
CBDD	SET	3,L	
CBE6	SET	4,(HL)	
DDCB05E6	SET	4,(IX+d)	
FDCB05E6	SET	4,(IY+d)	
CBE7	SET	4,A	
CBE0	SET	4,B	
CBE1	SET	4,C	
CBE2	SET	4,D	
CBE3	SET	4,E	
CBE4	SET	4,H	
CBE5	SET	4,L	
CBEE	SET	5,(HL)	
DDCB05EE	SET	5,(IX+d)	
FDCB05EE	SET	5,(IY+d)	
CBEF	SET	5,A	
CBE8	SET	5,B	
CBE9	SET	5,C	
CBEA	SET	5,D	
CBEB	SET	5,E	
CBEC	SET	5,H	
CBED	SET	5,L	
CBF6	SET	6,(HL)	
DDCB05F6	SET	6,(IX+d)	
FDCB05F6	SET	6,(IY+d)	
CBF7	SET	6,A	
CBF0	SET	6,B	
CBF1	SET	6,C	
CBF2	SET	6,D	
CBF3	SET	6,E	
CBF4	SET	6,H	
CBF5	SET	6,L	
CBFE	SET	7,(HL)	
DDCB05FE	SET	7,(IX+d)	
FDCB05FE	SET	7,(IY+d)	
CBFF	SET	7,A	
CBF8	SET	7,B	
CBF9	SET	7,C	
CBFA	SET	7,D	
CBFB	SET	7,E	
CBFC	SET	7,H	
CBFD	SET	7,L	
CB26	SLA	(HL)	Shift Operand Left
DDCB0526	SLA	(IX+d)	Arithmetic
FDCB0526	SLA	(IY+d)	
CB27	SLA	A	
CB20	SLA	B	
CB21	SLA	C	
CB22	SLA	D	
CB23	SLA	E	
CB24	SLA	H	
CB25	SLA	L	

OBJ CODE	SOURCE STATEMENT		OPERATION
CB2E	SRA	(HL)	Shift Operand Right Arithmetic
DDCB052E	SRA	(IX+d)	
FDCB052E	SRA	(IY+d)	
CB2F	SRA	A	
CB28	SRA	B	
CB29	SRA	C	
CB2A	SRA	D	
CB2B	SRA	E	
CB2C	SRA	H	
CB2D	SRA	L	
CB3E	SRL	(HL)	Shift Operand Right Logical
DDCB053E	SRL	(IX+d)	
FDCB053E	SRL	(IY+d)	
CB3F	SRL	A	
CB38	SRL	B	
CB39	SRL	C	
CB3A	SRL	D	
CB3B	SRL	E	
CB3C	SRL	H	
CB3D	SRL	L	
96	SUB	(HL)	Subtract Operand from Acc
DD9605	SUB	(IX+d)	
FD9605	SUB	(IY+d)	
97	SUB	A	
90	SUB	B	
91	SUB	C	
92	SUB	D	
93	SUB	E	
94	SUB	H	
95	SUB	L	
D620	SUB	n	
AE	XOR	(HL)	Exclusive 'OR' Operand and Acc
DDAE05	XOR	(IX+d)	
FDAE05	XOR	(IY+d)	
AF	XOR	A	
A8	XOR	B	
A9	XOR	C	
AA	XOR	D	
AB	XOR	E	
AC	XOR	H	
AD	XOR	L	
EE20	XOR	n	

Example Values

nn	EQU	584H
d	EQU	5
n	EQU	20H
e		30H

DEMONSTRATION PROGRAMS

9.5.1 PROGRAM 1

```
5 CUROFF
10 FORI=1TO50:NEXTI:SCREEN1
20 FORI=1TO18STEP2
30 CIRCLE(0,0),I,1
40 CIRCLE(239,0),I,1
50 CIRCLE(0,63),I,1
60 CIRCLE(239,63),I,1
70 NEXTI
80 CHAR2
90 LOCATE4,1
100 PRINT"DVW"
110 CHAR1
120 LOCATE11,1
130 PRINT"Microelectronics"
140 LOCATE12,2
150 PRINT"presents"
160 CHAR4
170 LOCATE9,3
180 PRINT"HUSKY SP"
190 FORI=1TO1100:NEXTI
200 CLS
210 CHAR3
220 LOCATE12,1
230 PRINT"HIGH RES"
240 LOCATE12,3
250 PRINT"GRAPHICS"
260 FORI=1TO999:NEXTI
270 CLS
280 FORI=0TO30STEP2
290 CIRCLE(120,31),I
300 NEXTI
310 FORI=1TO31STEP2
320 CIRCLE(120,31),I
325 SOUNDI,100
330 NEXTI
340 FORI=31TO1STEP-2
350 CIRCLE(120,31),I,0
355 SOUNDI,100
360 NEXTI
370 FORI=30TO0STEP-2
380 CIRCLE(120,31),I,0
390 NEXTI
392 CLS
395 FORI=1TO100:NEXTI
400 LINE(99,63-41)-(126,63-27),1,BF
402 LINE(96,63-44)-(133,63-24),1,B
403 LINE(96,63-44)-(92,63-48)
404 LINE(133,63-44)-(129,63-48)
405 LINE(96,63-24)-(92,63-28)
406 LINE(92,63-48)-(129,63-48)
407 LINE(92,63-48)-(92,63-28)
408 PSET(130,63-28),(130,63-20),(130,63-22),(130,63-24)
409 LINE(131,63-24)-(131,63-19)
410 LINE(98,63-24)-(98,63-10)
411 LINE(94,63-26)-(94,63-14)
412 LINE(127,63-24)-(127,63-14)
413 LINE(98,63-19)-(131,63-19)
414 LINE(98,63-17)-(94,63-21)
```

```
415 LINE(31,63-I)-(127,63-I)
416 LINE(94,62-21)-(127,63-21)
417 FORI=1TO999:NEXT
418 LINE(100,65-40)-(125,63-28),,BF
419 CHAR0
420 FORJ=1TO30
421 LOCATE18,4
422 PRINT"DVW"
423 LINE(101,62-30)-(124,63-30)
425 NEXTJ
426 LINE(99,62-41)-(126,63-27),,BF
470 FORI=1TO250:NEXTI
480 CLS
490 FORI=0TO62
500 LINE(56,I)-(183,I)
505 SOUND64-I,100
510 NEXTI
520 FORI=0TO63
530 LINE(119,63)-(119-I,0),I+1
540 LINE(120,63)-(120+I,0),I+1
550 NEXTI
560 FORI=0TO62
570 LINE(119,62)-(56,1),I+1
580 LINE(120,63)-(183,1),I+1
590 NEXTI
595 FORI=1TO500:NEXTI
600 FORI=0TO62
610 LINE(119,63)-(56,63-I)
620 LINE(120,63)-(183,63-I)
630 NEXTI
640 FORI=0TO62
650 LINE(119,63)-(56+I,0)
660 LINE(120,63)-(183-I,0)
670 NEXTI
680 FORI=63TO0STEP-1
690 LINE(56,63-I)-(183,63-I),0
695 SOUND64-I,100
700 NEXTI
720 CLS
730 CHAR1
740 LOCATE3,1
750 PRINT"5 Different character sets"
765 LINE(16,11)-(224,11)
767 FORI=1TO200:NEXTI
770 CHAR4
780 LOCATE8,3
790 GOSUB1000
800 LOCATE8,3
810 GOSUB1020
820 CHAR7
830 LOCATE12,7
840 GOSUB1000
850 LOCATE12,7
860 GOSUB1020
870 CHAR5
880 LOCATE8,7
890 GOSUB1000
900 LOCATE8,5
910 GOSUB1020
920 CHAR3
```



```
930 LOCATE12,7
940 GOSUB1000
950 LOCATE12,5
960 GOSUB1020
970 CHAR0
980 LOCATE17,4
990 GOSUB1000
999 GOTO10
1000 PRINT"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
1010 FORI=1TO999:NEXTI
1015 BEEP
1020 RETURN
1030 PRINT"
1040 RETURN
```

9.5.2 **PROGRAM 2**

```
10 CLEAR:CUROFF:SCREEN1
20 CIRCLE(120,31),30
30 LINE(119,3)-(119,6)
40 LINE(121,3)-(121,6)
50 LINE(145,31)-(148,31)
60 LINE(120,56)-(120,59)
61 LINE(92,31)-(95,31)
70 PSET(134,7),(144,17),(144,45),(134,55),(106,55),(96,45),(96,17),(106,7)
71 T$=TIME$:MI=VAL(MID$(T$,4,2)):HR=VAL(LEFT$(T$,2)):SC=VAL(RIGHT$(T$,2))
72 AM=MI*6+270:IFAM>359THENAM=AM-360
73 AH=HR*30+270:IFAM>359THENAH=AH-360
74 XM=FIX(23*COS(AM*PI/180)):YM=FIX(23*SIN(AM*PI/180))
75 XH=FIX(15*COS(AH*PI/180)):YH=FIX(15*SIN(AH*PI/180))
80 AG=SC*6+270:IFAG>359THENAG=AG-360:TI=500
90 IFAG>359THENAG=AG-360
100 TH=AG*PI/180
110 LINE(120,31)-(120+XS,31+YS),0
120 YS=FIX(25*SIN(TH)):XS=FIX(25*COS(TH)):LINE(120,31)-(120+XS,31+YS)
121 LINE(120,31)-(120+XM,31+YM)
122 LINE(120,31)-(120+XH,31+YH)
123 CHAR1,1:LOCATE1,1:PRINT" ";TIME$;" "
150 SOUND100,10
155 IFAG=270THENGOSUB200
160 TU=VAL(RIGHT$(TIME$,1))
170 IFTI=TUTHEN160
180 TI=TU:AG=AG+6:GOTO90
200 AM=AM+6
205 IFAM>359THENAM=AM-360
210 TH=AM*PI/180
220 LINE(120,31)-(120+XM,31+YM),0
230 XM=FIX(23*COS(TH)):YM=FIX(23*SIN(TH)):LINE(120,31)-(120+XM,31+YM)
240 IFAM<>270THENRETURN
250 AH=AH+30
255 IFAH>359THENAH=AH-360
260 TH=AH*PI/180
270 LINE(120,31)-(120+XH,31+YH),0
280 XH=FIX(15*COS(TH)):YH=FIX(15*SIN(TH)):LINE(120,31)-(120+XH,31+YH)
290 RETURN
```


9.5.3 **PROGRAM 3**

```
5 CUROFF
10 SCREEN1
20 CHAR4:LOCATE4,1:PRINT"HUSKY"
30 CHAR2,1:LOCATE2,3:PRINT"HUNTER"
35 LINE(15,26)-(14,38),1,B
40 CHAR1,0:LOCATE1,4:PRINT" BIG CAPABILITY"
50 PRINT" small PACKAGE."
60 LINE(0,0)-(125,63),1,B
70 CHAR0,0:LOCATE27,1:PRINT"The  worlds  most "
80 LOCATE27,2:PRINT"advanced hand held"
90 LOCATE27,3:PRINT"portable   micro ,"
91 LOCATE27,4:PRINT"HUSKY  HUNTER  is "
92 LOCATE27,5:PRINT"CP/M  compatible ,"
93 LOCATE27,6:PRINT"has upto  208K  of"
94 LOCATE27,7:PRINT"ram  and  powerful"
95 LOCATE27,8:PRINT"communications. "
100 INCHR"",A:SCREEN0
110 KEYOFF:CLS
120 PRINT"          A          B          C          D   "
130 OPCHR15,4,0
140 PRINTCHR$(124);:OPCHR15,39,0:PRINTCHR$(124);
150 OPCHR15,12,0,124,124
160 OPCHR15,21,0,124,124
170 OPCHR15,30,0,124,124
180 FORI=1TO7
190 OPCHR15,2,I
200 PRINTSTR$(I);CHR$(124);
210 NEXT
220 OPCHR15,4,1:PRINT"  [illegible]4.1[illegible]   45.8[illegible]   [illegible]98.4[illegible]   [illegible]98.11";
230 OPCHR15,4,2:PRINT"  54[illegible].72                [illegible]";
240 OPCHR15,4,3:PRINT"                     54[illegible]8.6[illegible]      [illegible]";
250 OPCHR15,4,4:PRINT"  [illegible]14.[illegible]   4[illegible].[illegible]6   85.64   [illegible]64.11";
260 OPCHR15,4,5:PRINT"            [illegible]4.8[illegible]             45.[illegible]";
270 OPCHR15,4,6:PRINT"                      8[illegible]4.64   407.45";
280 OPCHR15,4,7:PRINT"  1[illegible]4.8[illegible]   [illegible]45.49                 ";
290 OPCHR15,0,0:PRINT"            [illegible]48[illegible].45   4[illegible].[illegible]6";
295 INCHR"",A:CLS
300 CIRCLE(44,[illegible]1),[illegible]1
310 LINE(44,[illegible]1)-(66,9)
320 LINE(44,[illegible]1)-(44,62)
330 LINE(44,[illegible]1)-(15,41)
340 CHAR[illegible]
350 LOCATE6,[illegible]:PRINT"41%"
360 LOCATE[illegible],1:PRINT"UK"
370 LOCATE5,6:PRINT"2[illegible]%"
380 LOCATE[illegible],[illegible]:PRINT"USA"
390 LOCATE1[illegible],[illegible]:PRINT"[illegible]%"
400 LOCATE1[illegible],[illegible]:PRINT"Europe"
410 CHAR4:LOCATE[illegible]4,1:PRINT"ALME [illegible]"
420 CHAR[illegible],1:LOCATE1[illegible],[illegible]:PRINT"Profit Distribution"
430 LINE(8[illegible],26)-([illegible],[illegible])
440 CHAR1,0:LOCATE1[illegible],4:PRINT"[illegible] Year"
450 LOCATE[illegible]8,5:PRINT"1[illegible]"
460 INCHR"",A:SCREEN[illegible]
500 SCREEN1
510 LINE([illegible])-([illegible]),1,B
520 FORI=[illegible]STEP[illegible]
530 PSET([illegible])
```

```
540 PSET(I+1,60),(I+3,60),(I+6,60),(I+7,60)
550 PSET(I+7,61),(I+4,61),(I+5,61)
560 NEXT
570 LINE(50,60)-(45,50)
580 LINE(45,50)-(45,46)
590 LINE(45,46)-(39,40)
600 LINE(39,41)-(34,46)
610 LINE-(37,49)
620 LINE-(42,44)
630 LINE(45,46)-(62,46)
640 PSET(67,47),(67,48),(64,49)
650 LINE(45,50)-(150,50)
660 PSET(151,49),(152,48),(152,47),(152,46)
670 LINE(152,46)-(172,46)
680 LINE(177,46)-(155,60)
690 LINE(177,46)-(194,40)
700 LINE(145,50)-(145,10)
710 LINE(145,10)-(192,40)
720 LINE(71,50)-(71,15)
730 LINE(71,15)-(45,46)
740 LINE(108,50)-(108,1)
750 LINE(108,1)-(71,15)
760 LINE(108,1)-(145,10)
770 LINE(61,24)-(78,20)
780 LINE(58,39)-(84,54)
790 LINE(98,11)-(115,7)
800 LINE(95,25)-(121,30)
810 LINE(92,42)-(127,38)
820 LINE(135,19)-(152,15)
830 LINE(132,24)-(158,29)
840 FOR I=64TO150STEP10
850 PSET(I,53)
860 NEXT
870 LINE(160,48)-(156,53)
880 PSET(155,51),(155,50),(154,49),(157,53),(158,53),(159,52)
890 PSET(158,48),(160,50)
891 PSET(37,18),(37,17),(36,16),(35,16),(34,16),(33,17)
892 PSET(138,16),(139,16),(140,16),(141,17)
893 CIRCLE(0,0),15
894 PSET(210,51),(210,50),(209,49),(208,49),(207,49),(206,50)
895 PSET(213,49),(212,49),(213,49),(214,50)
900 CHAR$(1:LOCATE20,3:"PRINT HMS BOUNTY"
910 LINE(185,7)-(255,15),,1,B
915 INCHR$:,A:SCREEN1:GOTO1
```


9.5.4

PROGRAM 4 'UNLOAD'

On the host computer type the following program into a text file with the name UNLOAD.HEX. Before the file can be used it has to be converted to a .COM file, to convert the file enter the following command:

>LOAD UNLOAD.HEX

This will create a file with the name UNLOAD.COM and can now be used to convert any file into INTEL HEX format. To convert a file enter the folowing:

>UNLOAD sourcefile filename.HEX

The first file 'sourcefile' is the name of any file required to be converted, the second file 'filename.HEX' will then contain the INTEL HEX version of the file.

The HEX format file will always be located at 100H, but if all transferred data is saved into new files, (with the SAVE n command) then this is not a problem.

A source code version of UNLOAD is available on request from Husky Computers or authorised agents, either as a listing or 8" IBM compatible disk.

UNLOAD PROGRAM

```
:20010000C33402000000000000000000000000000000000000000000000000000000000000E6
:20012000000000000000000000000000000000000000000000000000000000000000000000BF
:20014000000000000000000000000000000000000000000000000000000000000000000000009F
:20016000000000000000000000000000000000000000000000000000000000000000000000007F
:20018000000000000000000000000000000000000000000000000000000000000000000000005F
:2001A000000000000000000000000000000000000000000000000000000000000000000000003F
:2001C000000000000000000000000000000000000000000000000000000000000000000000001F
:2001E000000000000000000000000000000000000044495343206572726F7224536F75726362
:20020000652F6F626A6563742066696C65206572726F722436F6E7665727469E672002E38
:20022002E2E202424436F6D706C6574656420E2E2E20242A060110100AFED52E5F1CD9C
:200240005E0401100216C0011C601EDB03A6D00FE20CC3D04115C000E0FCD0503CCC3DB6
:2002600041 1C6010E13CD05011C6010E16CD05003CCC4F04111402 0E09CD0500112402 3F
:20028000E09CD05011B0000E1ACD050011 5C000E14CD0500B7C41303AF32C5013AC50151
:2002A000FE802BE15F3C32C501160021 8000197ECD9D03210C0106003AC3014F09722373D7
:2002C0000C0C79FE342B0632C301C39D02AF32C301CDC403CDE90321C4017EFE012B103612
:2002E000012103011143010140 00EDB0C39D02360021030111B3010140 00EDB01143010E12
:200300001ACD050011C6010E15CD0500B7C44F04C39D02210C010603AC3014F09FE342B10
:20032000063630233C1B00CDC403CDE9033AC401F5FE012805114301180311B3012103014 7
:2003400014000EDB0F1FE012B0C3E0132C401210C013E001BC7114301 0E1ACD050011C6F4
:20036000010E15CD0500B7C44F04067F3E00214401772310FC060C2144013E30772310FC5F
:20038000215001 3E1A7711C6010E15CD0500B7C44F0411C6010E10CD0500C34604F5E60FC2
:2003A000FE0AFAA903C6371B02F6305FF1CB1FCB1FCB1FCB1FE60FFE0AFAC003C63757C983
:2003C000F63057C92AEE01111A001922EE01E57CCD9D037A3206017B320701E17DCD9D0368
:2003E0007A320B017B320901C921040106 1E110000AF7E4F237E23FE40FAFE03D607E60F22
:20040000B35F17171717F57BE60F5F79FE40FA1304D607E60FB257F13E008A5710D77BE613
:200420000F5F7A17171717E6F0B3ED44CD9D037A3240017B3241013E0D324201C911FB01DF
:20044000CD5B04C3000011 2502CD5B04C3000011F001CD5B04C300000E09CD0500C906EAFC
:20046000AF21030177 2310FC1124020E09CD0500210301363A2336312336412 1E60022EE12
:0904B0000121 0A013E0772377C7
:00000001FF
```


This page intentionally left blank.

9.6 KEYBOARD MEMORY MAP

Upper legend	Code	Code	Lower legend
!	65278	65334	1
@	65271	65327	2
#	65279	65335	3
$	65272	65328	4
%	65273	65329	5
^	65266	65322	6
&	65274	65330	7
*	65267	65323	8
(	65275	65331	9
)	65268	65324	0
_	65277	65333	-
+	65270	65326	=
↑	65276	65332	↑
ESC	65269	65325	BRK
PWR	65379	65380	HLP
Q	65292	65348	q
W	65285	65341	w
E	65293	65349	e
R	65286	65342	r
T	65287	65343	t
Y	65280	65336	y
U	65288	65344	u
I	65281	65337	i
O	65289	65345	o
P	65282	65338	p
]	65291	65347	[
←	65284	65340	←
LBL	65290	65346	INS
→	65283	65339	→
ENTER	65297	65353	
CTL	65306	65362	FN
A	65299	65355	a
S	65307	65363	s
D	65300	65356	d
F	65301	65357	f
G	65294	65350	g
H	65302	65358	h
J	65295	65351	j
K	65303	65359	k
L	65296	65352	l
:	65305	65361	;
"	65298	65354	'
↓	65304	65360	↓
DEL	65311	65367	BS
⇧	65320	65376	⇧
Z	65313	65369	z
X	65321	65377	x
C	65314	65370	c
V	65315	65371	v
B	65308	65364	b
N	65316	65372	n
M	65309	65365	m
SPACE	65317	65373	SPACE
<	65310	65366	,
>	65319	65375	.
?	65312	65368	/
\	65318	65374	TAB
⇧	65320	65376	⇧

9.7 MEMORY LOCATIONS

Location (Decimal)	Location (Hex)	Name	Function
55043	D703	JMP WARMSTART	CP/M BIOS jump table
55046	D706	JMP CONSTAT	
55049	D709	JMP CONIN	
55052	D70C	JMP CONOUT	
55055	D70F	JMP LIST	
55058	D712	JMP PUNCH	
55061	D715	JMP READER	
55296-55424	D800-DF80	VERSCRN	Virtual screen 80 x 24 characters D800 is the top left hand corner.
57344-61439	E000-EFFF		Operating system area.
61440-62463	F000-F3FF		I/O Buffers.
62981-63061	F605-F655	USER	This Ram area is left free by the operating system and is vailable for use as user machine code routines.
63168-63359	F6C0-F77F		O/S Stacks.
63360	F780	IPFLAG	Logical keyboard flag: 0 = HUNTER keyboard/LCD screen 128(80H) = Serial I/O port
63364	F784	VECTOR	HELP vector address. The key is trapped by the keyboard softwear and is vectored to this location.
63415	F7B7	APHRO	Flag to indicate how HUNTER was powered up: 0 = Keyboard 1 = Serial port 2 = Clock
63416	F7B8	NYMPHO	If the standard power down routine is used then setting this byte non-zero will inhibit operation of the power off key. **NOTE:** This byte is cleared on power up.
63419	F7BB	FOREVER	Setting non-zero prevents automatic switch off after time-out.

Location (Decimal)	Location (Hex)	Name	Function
63464	F7E8	TENTHSEC	Time memory 1/10 sec.
63465	F7E9	UNITSEC	" " unit sec.
63466	F7EA	TENSEC	" " tens sec.
63467	F7EB	UNITMIN	" " unit mins.
63468	F7EC	TENMIN	" " tens mins.
63469	F7ED	UNITHRS	" " unit hours.
63470	F7EE	TENHRS	" " tens hours.
63471	F7EF	UNITDAY	" " unit days.
63472	F7F0	TENDAY	" " tens days.
63473	F7F1	UNITMTH	" " unit month.
63474	F7F2	TENMTH	" " tens month.
63475	F7F3	UNITYR	" " unit years.
63476	F7F4	TENYR	" " tens years.
63480	F7F8	TXSPEED	Set serial O/P baud rate: 1 = 75 4 = 300 7 = 1800 2 = 110 5 = 600 8 = 2400 3 = 150 6 = 1200 9 = 4800
63481	F7F9	RXSPEED	Set serial I/P baud rate (same as (TXSPEED).
63482	F7FA	CTSAF	The CTS enable flag: 0 = No 1 = Yes
63483	F7FB	DTRAF	The DTR enable flag: 0 = No 1 = Yes
63484	F7FC	RTSAF	The RTS enable flag: 0 = No 1 = Hold 2 = True
63485	F7FD	DSRAF	The DSR enable flag: 0 = No 1 = Yes
63486	F7FE	DCDAF	The DCD enable flag: 0 = No 1 = Yes
63487	F7FF	TXPTY	Transmit parity flag: 0 = None 1 = Odd 2 = Even
63488	F800	RXPTY	Receive parity flag: 0 = None 1 = Odd 2 = Even

Location (Decimal)	Location (Hex)	Name	Function
63489	F801	TXPROT	Transmission protocol: 0 = None 1 = XON/XOFF 2 = ETX/ACK 3 = ACK/NAK 4 = SYSTIME
63490	F802	LFACT	Line feed active or not: 0 = No 1 = Yes
63491	F803	NULAF	Number of NULLs following CR/LF: 0 = None 1 = Two 2 = Five 3 = Ten 4 = Twenty
63492	F804	ECHOF	The serial output echo flag: 0 = No echo (Full Duplex) 1 = Output echoed to HUNTER's screen
63493	F805	RXPROT	Receive protocol (as TXPROT).
63494/63496	F807/F808	CTSVECT	CTS interrupt vector. These locations contain an address for the CTS input.
63502	F80E	ESCODE	Escape code. Used when escaping from fully running code. Used to interrupt execution of user programs when Basic auto-start is in use. ASCII characters are stored in Decimal form, most significant digit at lowest memory location.
63507	F813	STARTFL	Flag to indicate immediate running of user Basic program. AAH = Immediate run.
63551/63552	F83F/ F840	ALIM	Address of the lower limit of Basic Symbol table End of free Memory.
63553/63554	F841/ F842	DEFLIM	Beginning of free memory after Basic source and arrays.

Location (Decimal)	Location (Hex)	Name	Function
63563	F84B	SERIG	This contains the ASCII code of a character to ignore when receiving serial data. If not required set to 80H (128 decimal).
63571	F853	RECLEN	This is a single byte defining maximum record length. (See section 6.7)
63572	F854	BUFLEN	This is a two byte number defining maximum block length. (See section 6.7)
63574	F856	SRFLG	This flag controls single or multiple records in a block. (See section 6.7)
63575	F857	RPFLG	A single byte flag used to control record padding. (See section 6.7)
63576	F858	EMFLG	Determines essential differences between 2780 and 3780 communications. (See section 6.7)
63577	F859	STXLFG	A single byte flag controlling use of intermediate STXs. (See section 6.7)
63578	F85A	RETRYS	A single byte flag controlling error handling by the protocol. (See section 6.7)
63579	F85B	SCFLG	A single byte flag controlling operation of space compression. (See section 6.7)
63584	F860	BARTYP	Select Bar code type 0=CODE 39 1=EAN8/13
63585	F861	TOPAGE	Maximum number of pages in HUNTER. Note: Bit 7 is set to indicate "RAM".
63889	F991	LSTBSN	CAPS lock 0=OFF 1=ON Note: must be 0 or 1
65266	FEF2	KEYBUF	112 bytes of keyboard RAM, see section 9.6, KEYBOARD MEMORY MAP.

Location (Decimal)	Location (Hex)	Name	Function
65378	FF62	SPELFLG	Keyboard changed flag 1=changed keyboard layout
63420	F7BC	COMERR	Location which maintains the error code for a communications error. This location is normally clean.
63581	F85D	RXTOAF	Receive timeout active (parameters as TXTOAF).
63582	F85E	TXTOAF	Transmit timeout active 0 = no timeout 1 = 10 sec timeout 4 = 40 sec timeout 2 = 20 sec timeout 5 = 50 sec timeout 3 = 30 sec timeout 6 = 60 sec timeout

This page intentionally left blank.

9.8

PORT ALLOCATIONS

HUNTER uses the NSC800 port map for all its I/O functions, including the keyboard, RS-232, etc. The following gives a list of the most useful ports which can be controlled from Basic using OUT or INP verbs or from machine code.

Address Decimal	Hex	Name	Description
1	1H	DSR	Bit seven is the DSR input bit. 1 = Inactive 2 = Active
2	2H	INPUTS	This port has a number of input functions: Bit 0 = Data in (inverted from RS-232 line) Bit 1 = DCD - " - Bit 2 = Power low warning 1 = Power O.K 0 = Power low Bit 3 = TXCLK - " - also the single input bit.
96	60H	ANGLE	Controls display viewing angle. Input value in the range 0-1FH
129	81H	V24OUT	Directly outputs to the V24 data line signal on bit 0. The output is voltage inverted i.e. 0 = +Ve 1 = -VE
130	82H	RTS	RTS output bit
131	83H	POWHLD	Power control bit 0 = 1 : HUNTER on bit 0 = 0 : HUNTER off
132	84H	INVCON	V24 inverter control bit 0 = 1: inverter on bit 0 = 0: inverter off
133	85H	DTR	DTR output bit

135	87H	AUXPWR	Turns on the auxilary power to pin 10 on the 'D' type connector 1 = Power on 0 = Power off
187	BBH	ICR/FC	NSC800 internal interrupt mask register. See NSC800 hand book.
224	E0H	PAGE	Memory paging register

SINGLE BIT INPUT PORT

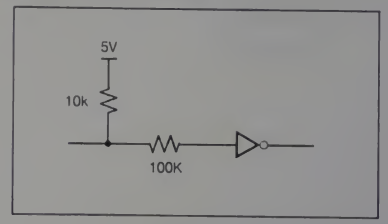
9.9

The single input bit has been implemented for use with devices such as bar-code wands, voltage free switch contacts, etc. It is provided via the 4 pin LEMO socket pin 2. Note should be made that it is shared with the TXCLK on the V24 interface (pin 15). It should not, therefore, be used at the same time as the communications when hardware handshaking is in use.

9.9.1 **CIRCUIT IMPLEMENTATION**

The input is configured as a protected, pulled up CMOS compatible line:

Fig 9.1 CIRCUIT LAYOUT



It has a logic 1 threshold of > 3.5V and a logic 0 < 1.5V. It is protected to ±25V.

The pull-up resistor enables the use of a switch to ground to provide the input.

No debounce circuitry is provided. If it is required then software must be written for this purpose.

This is bit 3 on I/O port 2. See 9.8

Connector as seen on Hunter

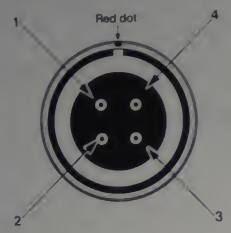


Fig 9.2 PIN-OUT OF LEMO CONNECTOR

PIN NO		SIGNAL
1	:	GROUND
2	:	TX CLK or input
3	:	Charger/reset
4	:	5V output (Wand option only)

9.9.3 **LEMO CONNECTOR**

The four pin LEMO plug is installed by aligning the red mark on the plug sleeve with the corresponding red mark on HUNTER's LEMO socket, the square pin on the plug body must also align with the key-way on the socket. The LEMO plug is then pushed firmly into HUNTER's socket until the outer sleeve of the plug is flush with the casing of HUNTER's LEMO socket.

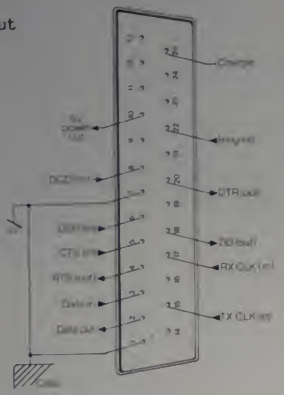
9.9.3.1 **Removal of LEMO Plug**

To remove the LEMO plug from HUNTER it is necessary to pull back the spring loaded outer sleeve before pulling the plug out of HUNTER's socket.

RS232 CONNECTOR

9.10 The 25 way D type connector is used for connecting standard serial devices using RS 232 to HUNTER.

9.10.1 **Connector Layout**



9.10.2 **Connector Type**

The connector is of the male type. It should only be used with standard female types.

9.10.3 **Connector Locking**

Two locking tabs are provided. These may be used to secure the mating connector securely.

Recommended clips are:

D53018 ITT Canon

For a full explanation of the use of these signals, see section 6 COMMUNICATIONS.

HEX DATA FORMAT

9.11 HUNTER provides the industry standard 'Intel Hex' data format compatible with most CP/M computers. .COM files may be converted to .HEX files with the UNLOAD utility in the host machine, and then 'pipped' out.

The data is split into blocks with leader and trailer information and the data in between. Each data block contains the memory locations into which the data is to be placed.

A single block of data appears as:-

:BBAAAAOODD.....DDCC

where BB = No of data bytes
AAAA = Address of block
OO = Null record
DD = Data byte
CC = Checksum

Each byte is composed of two ASCII hexadecimal characters (0-9 and A-F)

The record starts with a colon which is detected by the receiving system. This is followed by a count of the data bytes, the starting address, a delimeter, the data itself and a checksum on the end.

The checksum is calculated as the twos complement of the modulo 256 addition of all the preceding bytes.

The transmission is ended with a block of zero length.

:0000000000

9.11.1 **ERROR MESSAGES**

When loading HUNTER the address of the information is checked to ensure that it has written to the useable address space of the physical memory. Also the checksum is checked. If either of these checks fails, then an error message is displayed.

* * Memory overflow * *

or

* * * Loading Error * * *

as appropriate.

Reception at the end of record gives:

* * Loading completed * *

The loading may be aborted at any time by pressing ESC.

The length of input data records is optional and is specified by the record up to 255 bytes.

9.11.2 HEX TRANSMISSION

The length of the record is fixed at sixteen data bytes. To start the transmission HUNTER awaits the ENTER key to ensure that all connections have been made. The data may be echoed onto the screen to verify active transmission. The transmission may be aborted by pressing ESC.

9.11.3 LOADING MACHINE CODE PROGRAMS

The Hex data format will load any object code into memory. This may then be executed using the CALL statement in Basic, or loaded into a file with the file manager.

9.12 ASCII TO EBCDIC CONVERSION

When using EBCDIC communications HUNTER provides ASCII to EBCDIC conversion as follows:

(all values are in Hex)

ASCII	CHAR	EBCDIC	ASCII	CHAR	EBCDIC
00	NULL	00	20	SPACE	40
01	SOH	01	21	!	5A
02	STX	02	22	"	7F
03	ETX	03	23	HEX	7B
04	EOT	37	24	$	5B
05	ENQ	2D	25	%	6C
06	ACK	2E	26	&	50
07	BEL	2F	27	'	7D
08	BS	16	28	(	4D
09	HT	05	29	)	5D
0A	LF	25	2A	*	5C
0B	VT	0B	2B	+	4E
0C	FF	0C	2C	,	6B
0D	CR	0D	2D	-	60
0E	SO	0E	2E	.	4B
0F	SI	0F	2F	/	61
10	DLE	10	30	0	F0
11	DC1	11	31	1	F1
12	DC2	12	32	2	F2
13	DC3	13	33	3	F3
14	DC4	14	34	4	F4
15	NAK	3D	35	5	F5
16	SYN	32	36	6	F6
17	ETB	26	37	7	F7
18	CAN	18	38	8	F8
19	EM	19	39	9	F9
1A	SUB	3F	3A	:	7A
1B	ESC	27	3B	;	5E
1C	FS	22	3C	<	4C
1D	GS	1D	3D	=	7E
1E	RS	1E	3E	>	6E
1F	US	1F	3F	?	6F

ASCII	CHAR	EBCDIC	ASCII	CHAR	EBCDIC
40	@	7C	60	\	79
41	A	C1	61	a	81
42	B	C2	62	b	82
43	C	C3	63	c	83
44	D	C4	64	d	84
45	E	C5	65	e	85
46	F	C6	66	f	86
47	G	C7	67	g	87
48	H	C8	68	h	88
49	I	C9	69	i	89
4A	J	D1	6A	j	91
4B	K	D2	6B	k	92
4C	L	D3	6C	l	93
4D	M	D4	6D	m	94
4E	N	D5	6E	n	95
4F	O	D6	6F	o	96
50	P	D7	70	p	97
51	Q	D8	71	q	98
52	R	D9	72	r	99
53	S	E2	73	s	A2
54	T	E3	74	t	A3
55	U	E4	75	u	A4
56	V	E5	76	v	A5
57	W	E6	77	w	A6
58	X	E7	78	x	A7
59	Y	E8	79	y	A8
5A	Z	E9	7A	z	A9
5B	[	00	7B	{	C0
5C		E0	7C		6A
5D	]	00	7D	}	D0
5E	^	5F	7E		A1
5F	_	6D	7F	DEL	07

9.13

FOUR OCTAVE SOUND RANGE

	Note	Frequency (Hz)	Pitch Parameter
	C	138.810	493
	D	146.830	439
	E	164.810	391
	F	174.610	368
	G	196.000	326
	A	220.000	290
	B	246.940	260
	C	261.630	244
	D	293.660	217
	E	329.630	193
	F	349.230	182
	G	329.000	162
	A	440.000	144
	B	493.880	128
Middle	C	523.250	121
	D	587.330	107
	E	659.260	95
	F	698.460	90
	G	783.990	80
	A	880.000	71
	B	987.770	63
	C	1046.500	59
	D	1174.700	52
	E	1318.500	46
	F	1396.900	43
	G	1568.000	38
	A	1760.000	34
	B	1975.000	30
	C	2092.000	28

9.14 This appendix describes the installation procedure for a number of common commercial CP/M programs. The purpose of the installation is to configure the program to the screen handling control codes within HUNTER.

The control codes used by the HUNTER screen are:

a)	Clear Screen	01H
b)	Home Cursor	02H
c)	Cursor Addressing	0FH, <x co-ord>,<y co-ord>
d)	Cursor Right	0CH
e)	Cursor Left	08H
f)	Cursor up	0BH
g)	Cursor down	0AH

The installation procedure is generally carried out on a standard CP/M desk-top machine prior to down-loading information, see Section 3.6

9.14.1 **MBASIC**

Standard disk-based Microsoft Basic can be down-loaded into HUNTER with no installation or modification.

The only file to transfer is MBASIC.COM. MBASIC may now be run.

9.14.2 **WORDSTAR VERSION 2**

The standard disk from MicroPro has the following files:

INSTALL.COM
WSU.COM
WSMSGS.OVR
WSOVLY1.OVR

The INSTALL program is used to configure the main Wordstar program WSU.COM or an existing Wordstar file (WS.COM).

The two overlay files, WSMSGS.OVR and WSOVLY1.OVR, do not need any alteration.

When using the INSTALL program, refer to the Wordstar manual for further information - INSTALL is faily self explanatory. The main features of the procedure are explained below:

1) Type INSTALL to load and run INSTALL on your desk-top unit.
2) Select the appropriate Wordstar file, e.g. WSU.COM.

3) Terminal MENU : select "Z" for "none of the above." This will be described later when patching in the exact controls.

4) Printer MENU : select your own type of printer - typically, selection "A" will be compatible with most types of printer.

5) Communications Protocol MENU : select "N", this will enable HUNTER's own communications package to handle any necessary protocol.

6) Driver MENU : select "L" for the standard CP/M list device.

7) You are now prompted for:

ARE MODIFICATIONS TO WORDSTAR NOW COMPLETE?:

Answer "N" for No, as it is now necessary to patch the codes for the terminal selection.

INSTALL will prompt with:

LOCATION TO BE CHANGED (0=END):0248
ADDRESS:0248H OLD VALUE:00H NEW VALUE:18

In the above example, location 0248 was entered and the New Value 18H entered. This procedure should be repeated for the following locations:

LOCATION (Hex)	New Value (Hex)	FUNCTION
0248	18	Screen Height (24 lines)
0249	50	Screen width (80 characters)
024A	01	Clear screen
024B	0F	Cursor positioning
0253	00	"
0258	00	"
025D	FF	"
025E	00	"
025F	00	"
0260	00	"
0264	00	"
02AA	00	Inhibit scrolling

After engineering these patches, terminate the procedure by entering 0 when the next location is requested.

CONFIGURING TYPICAL CP/M PROGRAMS

After confirming the selections and exting from INSTALL, the three files:

WS.COM
WSMSGS.OVR
WSOVLY1.OVR

are ready to be loaded and run, see Section 3.6 for details of the procedure.

9.14.3 **SuperCalc Version 1**

The standard disk from Sorcim has the following files:

INSTALL.COM
SC.COM
SC.OVL
SC.HLP

The INSTALL program is used to configure the main program SC.COM. The two other files SC.OVL and SC.HLP do not need any alteration.

It will be helpful to refer to the SuperCalc manual when using INSTALL.

However, below is set out the response to each menu choice to install for the HUNTER, after running the "INSTALL.COM" program.

a) Terminal Selection Menu

No automatic installation is provided for HUNTER. Type in 'x' (not specified on the screen) for manually setting up the various options.

A second menu is displayed showing options A-F. Each option should be configured as described.

A. EDIT SCREEN CONTROLS

Another menu is displayed. Choose each option in turn and modify as prompted until the following is displayed:

1 Clear Screen : 1,01
2 Clear to EOL : Unconfigured
3 Home cursor : Unconfigured

Return to the main selection and run by pressing x.

B. EDIT ATTRIBUTE DATA

No configuring is necessary. Each option should be:

1 Set (start) Cursor attribute : Unconfigured
2 Clear (end) Cursor attribute : Unconfigured
3 Set (start) Border attribute : Unconfigured
4 Clear (end) Border attribute : Unconfigured

C. EDIT INPUT KEYS

1 Keyboard lead-in character : No lead-in character
2 Key to input for -up : K (Control K)
3 Key to input for -down : J
4 Key to input for -left : H
5 Key to input for -right : L
6 Key to input for Help : No help key

D. EDIT GOTO XY, PRINTER INIT. STRING

1 Printer init. string

leave unconfigured

2 Goto XY routine (for cursor addressing)

type in the following string of bytes:

D5, 3E, OF, CD, 7D, 01, D1, D5, 7B, D6, 20, F6, 80,
CD, 7D, 01, D1, 7A, D6, 20, F6, 80, C3, 7D, 01

E. EDIT MISCELLANEOUS DATA

0	Display Border Character	-7Ch
1	No. of video lines (down)	-12
2	No. of video columns (across)	-40
3	No. of printer lines	-66
4	No. of printer columns	-132
5	Dumb terminal cursor brackets Left = "<" Right = ">"	
6	CPU frequency (in 1000s Hz)	-4000
7	Baud rate used (for delays)	-12
8	No. of CRT attribute	-0
9	Use printer status (in BIOS) (1 = yes, 0 = no)	-1

F. EDIT TERMINAL NAME

Enter name = Husky HUNTER

Then type X to finish editing data, and A to save SuperCalc on disk.

The files:

SC.COM
SC.OVL
SC.HLP

are now complete.

9.14.4 **Loading dBASE Programs**

The well known data base package, dBASE II, can be installed to run on the HUNTER. Described below is the sequence of responses to the install program to give full functions to the software package.

The version described is V2.3B, which is supplied as 13 individual files. They are:

DBASE.COM	DBASEMOD.OVR
DBASEMSG.COM	DBASEMSC.OVR
DBASEMAI.OVR	DBASERPG.OVR
DBASEAPP.OVR	DBASESRT.OVR
DBASEBRO:OVR	DBASETTL.OVR
DBASEJOI.OVR	DBASEUPD.OVR
	INSTALL.COM

The first three files are vital to any dBASE program, but the rest are separate overlay files each supporting a particular function.

For example, if a sort facility is required, then the DBASESRT.OVR file will be required on the HUNTER.

9.14.5 **Installing dBASE**

On your host CP/M machine, type:

INSTALL

to run the install program.

The first question will ask whether you require full screen operations.

Type 'Y'

A menu of diferent types of VDU will then appear.

Select 'Z'

for user supplied terminal commands.

The next question refers to the type of data to be entered. All data will be in hex, so answer accordingly.

1. DELETE A CHARACTER SEQUENCE

   The correct sequence is 03,08,20

2. DIRECT CURSOR POSITIONAL SEQUENCE

   Column number position = 02
   Line number position = 03
   Constant bias = 00

   The skeleton is 0F,00,00

3. CLEAR AND HOME SCREEN COMMANDS

   01

4. BRIGHT/STD VIDEO COMMANDS

   None

5. DIM/REVERSE VIDEO COMMANDS

   None

6. INITIALISATION SEQUENCE

   None

7. EXIT SEQUENCE

   01

8. RESET TO STANDARD VIDEO MODE

   None

9. ALTER SCREEN SIZE

Screen width = 40
Number of lines = 08

9.14.6 **Completion of Installation**

To complete installing, default the macro character to "&" and key-in a space for no error dialogue.

The CP/M operating system is version 2.2., therefore:

Enter "A"

to complete the installation. The above files are now ready for installation.

9.15 **Read Console Buffer**

CP/M call 10, Read Console Buffer, has a range of additional facilities.

The read buffer reads a line of edited console input into a buffer addressed by registers DE. Console input is terminated when either the buffer overflows, or on control J or control M.

The Read buffer is of the form:

DE	+0	+1	+2	+3	+4	+5	.....	+N
	mx	nc	c1	c2	c3	c4	.....	??

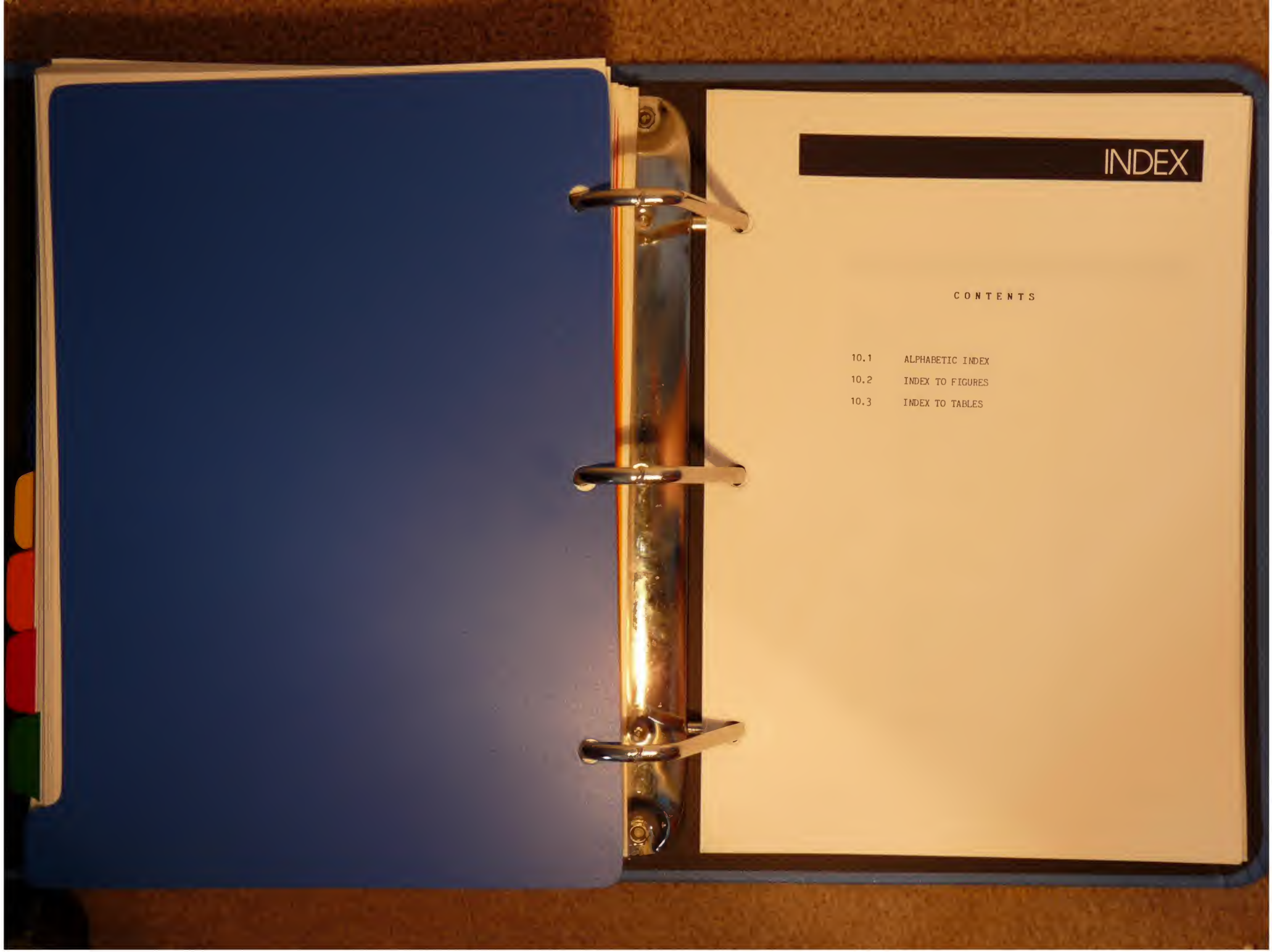
where mx is maximum number of characters the buffer will hold (1 to 255).

nc is the number of characters read (set by DEMOS on return). nc is followed by C1, C2, etc., the characters read from the console.

If nc < mx then uninitialised positions follow the last character (represented above by ?).

A number of control functions are recognised during line edit:

Control C	reboots system if at start of line
Control E	causes physical end of line
Control H	backspace one character position
Control J	terminate input
Control M	terminate input
Control R	retype current line after new line
Control X	backspaces to beginning of current line


INDEX
CONTENTS
10.1 ALPHABETIC INDEX
10.2 INDEX TO FIGURES
10.3 INDEX TO TABLES

10.1

ALPHA INDEX

10.2

INDEX TO FIGURES

10.3

INDEX TO TABLES